Aalto University
School of Science
Degree Programme in Computer Science and Engineering

Lari Sinisalo

Logical segmentation and labeling of PDF documents

Master's Thesis
Espoo, April 24, 2014

Supervisor: Professor Aristides Gionis
Advisor: Rami Hänninen Lic.Sc. (Tech.)

Aalto University
School of Science
Degree Programme in Computer Science and Engineering

ABSTRACT OF
MASTER'S THESIS

Author: Lari Sinisalo

Title:
Logical segmentation and labeling of PDF documents

Date:	April 24, 2014	**Pages:**	74
Major:	Information and Computer Science	**Code:**	T-61
Supervisor:	Professor Aristides Gionis		
Advisor:	Rami Hänninen Lic.Sc. (Tech.)		

Electronic documents are distributed in various different formats, which concentrate on different aspects, such as editability or precise graphical control. PDF documents do not contain any logical textual structure, and therefore recognizing even a single line of text in a PDF document is a non-trivial task.

The conversion of PDF documents into structured formats requires the reconstruction of the documents' logical structure. In this thesis, the selected output format is a structured combination of HTML and CSS. Paragraphs, lists and tables are the logical components that are of particular interest to this research.

This thesis presents a modular, general purpose system for reconstructing a logical structure in PDF documents. The development of a general purpose system is still an unsolved problem, as logical reconstruction systems tend to be specialized in specific classes of documents. This issue is addressed by devising a modular, extensible system based on basic properties of human perception.

The implemented system is compared to other logical reconstruction systems, and additionally PDF reader and text extraction software. The implemented system is strictly rule-based and procedural, which is known to limit its accuracy and to be a disadvantage compared to the more advanced methods used in specialized systems.

The devised system still requires improvements to get close to the accuracy of the specialized systems, however the selected approach is very promising. Future work and improvements are considered at the end of this thesis.

Keywords: PDF document, reconstruction, logical structure, reverse-engineering, modular

Language: English

Aalto-yliopisto
Perustieteiden korkeakoulu
Tietotekniikan koulutusohjelma

DIPLOMITYÖN
TIIVISTELMÄ

Tekijä:	Lari Sinisalo		
Työn nimi:	PDF-dokumenttien looginen segmentointi ja luokittelu		
Päiväys:	24. huhtikuuta 2014	**Sivumäärä:**	74
Pääaine:	Tietojenkäsittelytiede	**Koodi:**	T-61
Valvoja:	Professori Aristides Gionis		
Ohjaaja:	Tekniikan lisensiaatti Rami Hänninen		

Sähköisten dokumenttien esittämiseen käytetään useita erilaisia formaatteja. Eri formaatit erikoistuvat eri käyttötarpeisiin, kuten muokattavuuteen tai tarkkaan graafiseen hallittavuuteen. PDF-dokumentit eivät välttämättä sisällä tietoa dokumentin loogisesta rakenteesta, eikä edes yksittäisen rivin tunnistaminen tekstistä ole täysin suoraviivaista.

Jotta PDF-dokumentteja voidaan muuttaa toisiin formaatteihin, täytyy tekstin looginen rakenne palauttaa eli rakentaa uudelleen. Tässä työssä loppuformaattina on käytössä rakenteellinen yhdistelmä HTML- sekä CSS-merkintäkieliä. Tältä kannalta kiinnostavia dokumentin loogisia osia ovat erityisesti tekstikappaleet, listat ja taulukot.

Tässä työssä esitellään PDF:n rakennetiedon palauttamiseen suunniteltu modulaarinen, yleiskäyttöinen järjestelmä. Loogisen rakennetiedon palauttamiseen käytetyt järjestelmät keskittyvät yleensä yksittäisiin dokumenttityyppeihin, eikä täydellistä yleiskäyttöistä järjestelmää ole vielä luotu. Työssä tämän ongelman ratkaisua lähestytään esittelemällä modulaarinen, helposti laajennettava järjestelmä, joka pohjautuu ihmisen havaintokyvyn perusominaisuuksiin.

Toteutettua järjestelmää verrataan muihin loogista rakennetietoa palauttaviin järjestelmiin, sekä PDF-dokumentteja esittäviin tai niiden tekstisisältöä käsitteleviin ohjelmiin. Järjestelmän toteutus on täysin sääntöpohjainen ja proseduraalinen, minkä tiedetään rajoittavan järjestelmän tarkkuutta ja heikentävän järjestelmää suhteessa erikoistuneempiin menetelmiin.

Työssä kehitetty järjestelmä vaatii vielä parantamista jotta sen tarkkuus yltäisi samalle tasolle kuin erikoistuneemmissa järjestelmissä, mutta valittu lähestymistapa on hyvin lupaava. Jatkokehitystä sekä parannuksia pohditaan työn lopussa.

Asiasanat:	PDF-dokumentti, rakenteen palauttaminen, looginen rakenne, takaisinmallinnus, modulaarisuus
Kieli:	Englanti

Acknowledgements

I wish to thank my supervisor Aristides Gionis and my instructor Rami Hänninen for their help in creating this thesis. Thanks also go to Esko Räty and Joni Vähämäki who helped polishing the final version of this thesis. I also wish to thank Documill Oy for providing an interesting subject to research.

Thanks for all the help and support you gave.

Espoo, April 24, 2014

Lari Sinisalo

Contents

Chapter 1

Introduction

Electronic documents are traditionally distributed in various different formats. Different document formats cater to different needs, such as representing content with high graphical and typographical quality or allowing editing the document while simultaneously seeing the visual result. The formats that allow precise control on document appearance do not necessarily preserve the original logical structure of the document.

Every human readable document has a physical structure that specifies what exists on a document page and where. Documents can also have a logical structure, which denotes what kind of logical components are on the pages and which parts of the physical structure belong to each component. Logical components include paragraphs, lists, tables and various other types. Not every document format has a logical structure.

Documents that are expressed in logically structured formats are relatively simple to express in other formats while preserving the layout and appearance. The formats used by various word processing applications are typically structured. At the opposite end, most documents written in the PDF format contain no logical structure, and drawing text is just one graphical operation among others. Even recognizing a single line of text is a non-trivial task in the general case.

The lack of logical structure becomes a problem when converting documents into other formats or trying to edit a document. The lack of logical structure means it is not known how a change would affect the existing content on the page, or how it should be presented in another format. The other format may support precise replication of the visual appearance of the original document, but such conversion would not necessarily create a well-formed document if having a logical structure is expected in the other format.

Recognizing and reconstructing the logical structure in non-structured documents is still an open question. A significant amount of research has been

put into optical character recognition of scanned documents and recognizing basic units such as lines or blocks of text. This is still only a starting point for reconstructing the actual logical structure of the document.

The existing research and methods for reconstructing the logical structure will be studied in this master's thesis. Utilizing the existing research and original ideas, a document reconstruction system will be devised, implemented and analyzed for this task.

In this master's thesis the focus is on studying the problem domain more closely, defining what actually is being solved, and implementing a system based on the introduced methods. The problem of reconstructing the structure of a document is split into subproblems following basic principles on how the human perceptive system works. This gives the benefit of being independent of any specific class of documents, and that the intermediate results are increasingly more structured from a human reader's point of view.

1.1 Problem statement

The problem being solved is recognizing paragraphs, lists and tables in the text of an input PDF document that lacks explicit structural information. This is a simple task for a human reader, but a difficult task for a computer. Even a simple document such as in figure 1.1 requires relatively complex algorithms.

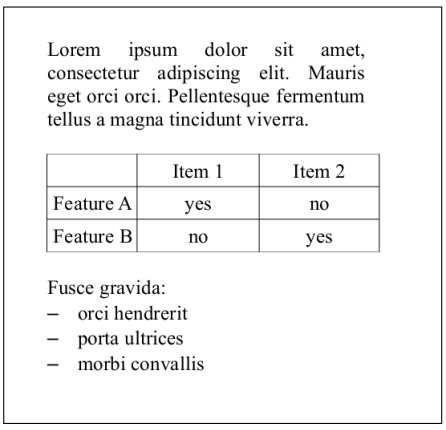


Figure 1.1: A simple document page with a single paragraph, table and a list.

This simple problem statement is ambiguous, as it relies on the reader's own understanding of what paragraphs, lists and tables are. The complexity

of the problem can be demonstrated by formalizing the problem statement, as follows.

1.2 Formalization

The following problem statement formalizes the problem in a way that depends on the human interpretation of the visual representation of the document content:

Input: A set of characters C. For each character $c_i \in C$, a description G_i of a set of points that form the exact position and shape of the character's glyph on a page. A set of indicator functions I, where each $\mathrm{I_k} \in I$ is a function from subsets of C to $\{0, 1\}$.

Output: A partitioning of C into sets C_j satisfying $\forall j \exists k(\mathrm{I_k}(C_j) = 1)$ (each set of characters is recognized by at least one indicator function).

The intuition behind this problem formalization is that each character is part of some logical component, which can be interpreted as a paragraph, list or a table. Each indicator function specializes in recognizing its own logical component type. Partitioning the characters into groups recognized by at least one indicator function gives one interpretation of the document structure.

This problem statement still omits the definition of paragraphs, lists and tables, and the problem is simply transformed into defining the indicator functions I which can recognize them. The benefit of this definition is that it modularizes the problem to be solvable once the necessary indicator functions are known.

Unfortunately these indicator functions rely on knowing the human interpretation for each case, which is not very practical. However, this still allows solving the problem approximately, and there are many methods for approximating human interpretation accurately and efficiently.

This formalization is used in the logical segmentation algorithm implemented in this thesis. The algorithm utilizes component interpreters to aid in segmentation of the page. These interpreters correspond to the indicator functions, as they try to interpret candidate logical components which are basically sets of characters. The identified components are then labeled as the recognized type, such as a list or a table.

1.3 Modularization

The implemented system is modularized into multiple components. The tasks to be done include text extraction from PDF documents, text line and block detection, logical segmentation and logical labeling. The overall process is illustrated in figure 1.2.

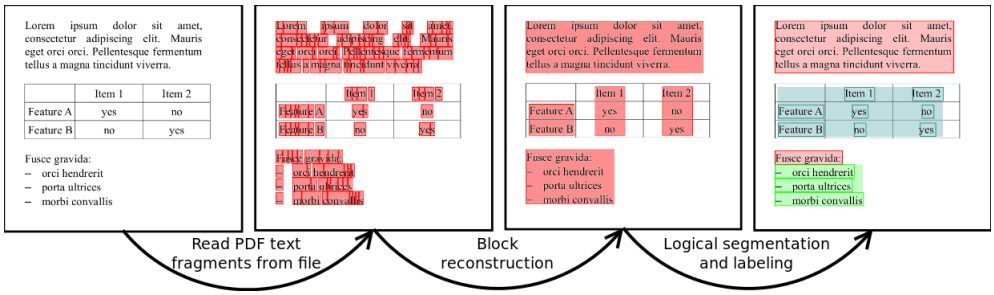


Figure 1.2: The overall process from a PDF file into structured output. Table component is shown with blue background and list component with green.

The text extraction step is performed by Documill Publishor, which is a server-side document processing software developed by Documill Oy. Depending on how the PDF document was constructed, the text fragments extracted from the PDF file can range from complete lines to single characters, so initially they lack all logical structure information. Documill Publishor and the input are described in chapter 3, Environment.

Detection of text lines and blocks is done using an algorithm inspired by Kruskal's minimum spanning tree algorithm, and other text block reconstruction algorithms that are reviewed as prior research. This gives an initial result that is comparable to the text selection capabilities in various PDF reader software. At this point the paragraphs in the text are usually well-formed and usable.

Logical segmentation and labeling are done in order to get further structure and identify more complex components such as lists and tables. The logical segmentation of the identified blocks of text is implemented as a variant of the basic XY-cut algorithm introduced in prior research in chapter 2, Background.

The segmentation process is guided by component interpreters, which handle the logical labeling and structuring of individual components like lists and tables. This approach is inspired by details in human structural perception discussed in the background chapter.

The reconstructed logical structure is known after logical segmentation and labeling. The result is then given to another component of Documill

Publishor and written into a structured format using HTML and CSS.

1.4 Structure of the thesis

This chapter was an introduction to the problem domain of the thesis. The next chapter, Background, will explore the problem domain and prior research further and provide background information essential to fully understanding the problem at hand. The environment in which the system will be built will be introduced in chapter 3, Environment.

The chapter 4, Methods, will present the methods that are used for reconstructing logical structure. Each component of the modularized system will be considered separately. The methods will be considered at a higher abstraction level, defining what they do and what are the main ideas that are used. The available information that can be utilized in the devised methods will also be defined more closely.

The chapter 5, Implementation, will explain how the methods introduced in the previous chapter are implemented. The chapter will consider the finer details behind how the methods work and what are the practical problems when applying them.

The system will be evaluated in chapter 6, Evaluation using test sets of real-world documents and simpler specialized test documents. The system will also be compared to available basic PDF viewer and text extraction software, and to other similar logical reconstruction systems.

The results from the evaluation and suitability of the chosen approach will be discussed in chapter 7, Discussion. Further improvements and future work will also be considered. The work will be summarized and concluded in the last chapter 8, Conclusions.

Chapter 2

Background

Understanding the problem domain is essential in creating a solution to the problem. This chapter provides an overview on how reconstructing the structure in a document is approached.

Electronically analyzing documents is a well researched problem, but fully reconstructing the structure is still an open question. Many successful methods have been designed for specific classes of documents, such as scientific reports, manuals or newspapers. It is not always clear how the ideas could be used when recognizing structure in other classes of documents.

More flexible methods can not assume that all input documents come from a single class of documents. The human perception system is studied in this chapter to find ways to recognize the structure of documents in a document class independent way.

It will be shown that extracting structured text from documents written in the Portable Document Format is close to recognizing structure after applying optical character recognition. Relevant prior research and existing implementations are studied to help choosing the right approaches for designing the system in development.

2.1 Definition of a document

A document may be defined in multiple ways. In this thesis a document is defined to be the visual presentation of any information that is distributed over one or multiple pages. This thesis itself is an example of a document meeting the definition.

Documents are generally very visual in nature, relying on normally functioning eyesight to fully comprehend. The presented algorithms will try to find what an average human reader perceives on a document page. As such,

the approach chosen in this thesis may not be suitable for documents targeted for non-human readers.

The main work in this thesis concentrates on the textual content of documents. Utilizing further information is considered at the end of this thesis.

2.1.1 Physical and logical structure

The structure of a document can be divided into physical and logical layers, which both give valuable information on how to interpret the document. These definitions follow the basic ideas introduced in a paper on a document logical restructuring system by Bloechle [2].

The physical structure is what there physically is on the document page when it is viewed. The perceived document page may be composed of paper and ink or bits and pixels, and these small units form larger structures that convey information to human readers. The exact way the physical structure is presented does not normally affect how the document is interpreted.

The connected components of the small units are usually the smallest relevant piece of information when considering how the document is perceived. A connected component is defined here as a set of small units, such as pixels or glyphs, which are grouped together according to some criteria. The criteria can vary depending on what the connected components are supposed to represent.

The individual glyphs on a document page can be grouped together into a connected component by joining glyphs together if the distance is below a certain threshold. A more complex set of criteria can be derived, for example, from the Gestalt principles that are discussed later in this chapter.

The logical layout tells what the physical layout components actually are. A set of glyphs may form a paragraph, table or list, which all are logical components. These components form more complex hierarchies, but only this basic set is considered in this thesis as the higher hierarchy levels are increasingly document-class specific.

When the logical structure is known, laying out a document page is a relatively simple task. Reverting this and going from document end presentation back to physical and logical structure is a much more difficult task when the information is not saved.

The logical layout is important for understanding how to interpret the contents of the page. Especially tables are hard to interpret without understanding their structure. In document formats that lack logical structure, reconstructing logical structure is required to recognize even the most basic textual units, such as words or lines of text.

2.2 Portable Document Format

The Portable Document Format (PDF) [21] is an excellent format for presenting any kinds of documents in a platform-independent way for human readers. It is a standardized format, readable on all kinds of devices from mobile phones to personal computers.

The documents expressed in the PDF format are easy to read for human readers. Automatically processing the contents is not as easy, as PDF is essentially a graphical format. There is no logical structure in the documents unless explicitly added by the writing software, and this is rarely the case.

Text in PDF is written using either of two text operators, *Tj* and *TJ*, as defined by the PDF specification. These operators draw glyphs to a specific position on a page, as defined by multiple layers of affine transformation matrices. The exact order in which text is written and details of how the affine transformations and operators are used depends on the document writer. The PDF format allows numerous equally valid ways of producing the same visual result, and therefore no structure can reliably be derived from how the text operators are used.

The strings of text given as arguments to the drawing operators may not represent anything useful such as complete lines or words. Especially spaces between words are problematic, as they are not visible to the end user. A significant number of PDF writers omit the spaces in text completely and instead use the various other ways to control where the text appears. Some writers even separate table columns by using the spaces in the text strings in combination with adjusting the spacing between words.

These non-semantic text drawing practices mean that the largest reliable units of text extracted from PDF documents are individual characters. This indicates that the problem of extracting structured text from PDF documents is quite close to the problem of extracting structured text from printed documents just after applying optical character recognition (OCR).

2.3 Human perception

Documents are generally created for human readers with a normally functioning visual system, so understanding how human readers perceive a document is vital for creating a general purpose document analysis method. In this thesis, perception is considered to include both purely visual processes in the visual system, and also the more sophisticated psychological processes relevant to fully understanding a document.

2.3.1 Gestalt principles

The Gestalt principles are basic guidelines that describe how human visual perception works. They describe especially how different visual objects are grouped together. Particularly interesting principles are the principles of proximity, similarity and closure.

The principle of proximity tells that objects close to each other tend to be perceived as grouped together. Proximity is relative, so the distance between elements in a group depends on the context.

Figure 2.1 illustrates the principle of proximity. The figure is interpreted as one square group of nine dots at the left, and three horizontal groups of three dots at the right. Furthermore, the three horizontal groups can also be interpreted as a group separate from the group at the left, creating a nested hierarchy.

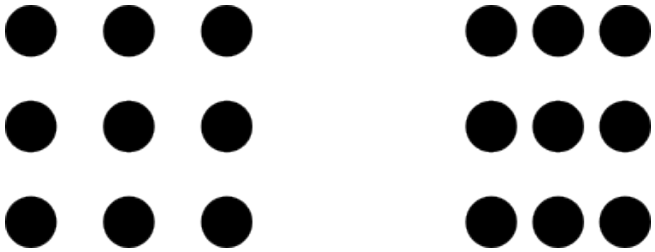


Figure 2.1: Gestalt proximity principle visualized. Human viewers usually perceive the dots on the left side as one square group of nine dots, and the dots on the right side as three horizontal groups of three dots.

The principle of similarity tells that similar objects are perceived as belonging together. Figure 2.2 shows nine objects of three different shapes. Each set formed by the same shape is perceived as a group, resulting in one group of circles, one group of stars and one group of pentagons.

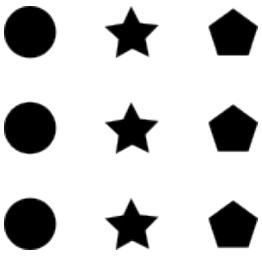


Figure 2.2: Gestalt similarity principle visualized. Each different set of similar objects is perceived as one group, giving a total of three groups.

The principle of closure states that incomplete objects can be perceived as complete. In figure 2.3 there are three segmented lines, which can still be perceived as complete lines.

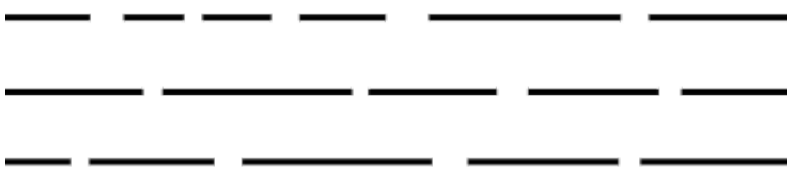


Figure 2.3: Gestalt closure principle visualized. The three horizontal sets of line segments are perceived as groups forming complete lines.

Other Gestalt principles discuss symmetry, good continuation and many other details. The Gestalt principles are further discussed in a book by Blake and Sekuler [1] and various other sources.

The human visual system is very complex, so the principles are not absolute truths and the perceived result may be affected by various other effects. There also may be multiple ways to interpret the same visual scene, so the results derived using the Gestalt principles may not be unique. Nevertheless, the principles give valuable insight into the human visual system.

2.3.2 Structural perception

In addition to small details, the human perception recognizes higher level constructs consistently. In a study on paragraphs [11], the psychological processes behind paragraph identification were studied with experiments. The study shows that even when words in non-indented text are replaced with nonsense words, the readers split the text into the same paragraphs quite consistently. This indicates that the paragraph is a real unit in the human perception system. Documents written for human readers most likely implicitly group the textual content into paragraphs and similar units.

One important question is whether the human perception processes the visual contents starting from local to global, global to local, or a combination of both. Local to global would mean starting from local, small features and proceeding to larger structures, and global to local means starting from the global view and moving to smaller details from that direction.

Many different methods for document analysis have been devised, many of them either local to global or global to local. Especially global to local methods usually have certain classes of cases where they do not work, like the recursive XY-cut which will be discussed later in this chapter. Local to

global methods might utilize some sort of higher level knowledge that they build while processing, which shifts them towards hybrid methods. Usually they, too, have specific classes of cases where they do not work either.

There are indications that the human visual system does not use a purely local to global or local to global approach. In a study on global and local processing [14], the human visual system is concluded to be more opportunistic, as the research finds problems with both pure approaches. Interestingly their explanation is based on structural instead of spatial properties, indicating that the same property might be relevant even in non-visual processing.

2.3.3 Perception in document analysis

Many interesting methods based on how human perception works have been devised. In one method [13], text lines were extracted from handwritten documents by using the physiology of vision and the Gestalt laws. Perceptual grouping by proximity and direction continuity were noted as especially useful.

The method concentrates on detecting text lines, instead of individual characters or words. It requires no prior knowledge on line orientation, unlike other methods mentioned in the work. This indicates perceptual methods can derive more useful information than methods that don't take human perception into account.

Another interesting method [6] is an approach for the visual segmentation of a document. The method simulates human visual system features at the retina level, and also uses the Gestalt theory and other approaches to form a concept of attention. These ideas are used to find the interesting areas on the document page. The authors also consider extending the method by taking psychological criteria into account, instead of only physiological perception. An important observation is that text lines are perceived as homogeneous blocks. This corresponds with the observation that paragraphs are a useful unit in the human perceptive system.

2.4 Prior research

Prior research particularly interesting for creating a modular system include methods that recognize blocks of text from a set of smaller units of text and methods that can be used to recognize further structure in the text. There are many methods for recognizing blocks of text, some of which will be reviewed here. To recognize further structure, a basic approach to segment

page contents is reviewed and used as a basis for a segmentation algorithm in chapter 4, Methods.

Further study on document structure recognition algorithms can be found for example in a study on the state of the art in structure recognition of scanned documents as of 2003 by Mao et al. [15]. Some categorization is done also by Bloechle [2]. Comprehensive surveys of table recognition methods and ideas were done by Embley et al. [7] and Zanibbi et al. [19].

2.4.1 Block reconstruction

A simple algorithm for recognizing blocks of text was introduced as a block segmentation method by Kieninger [10]. The method specializes in documents with tables and especially separating tightly packed columns from each other. The method works by starting at a fragment of text and iteratively expanding to the left and right and the above and below lines until the block is fully found. This initial approach does not work if there are vertical gaps in a paragraph, and these cases are corrected using additional rules.

The method relies on knowledge or estimation of the next and previous lines, and also requires full words to be known or estimated so that spaces can be compared to avoid splitting at vertical gaps. In the case of PDF input spaces are not known as shown earlier in this chapter. Some ideas from this algorithm are used in the implemented system in table interpretation.

A method for finding blocks of text in PDF documents was described for converting documents into a structured XCDF format [3]. The method creates a layer for each text rotation, and processes them individually as horizontal text. The text fragments are merged horizontally using a dynamic distance threshold, tokenized into words, numbers and other textual primitives, and then merged horizontally into lines. The lines are merged into blocks by using a dynamic distance threshold and avoiding merging non-connected lines of text. In the case of justified lines the text lines may be oversegmented as the spacing can vary a lot, so these are retroactively corrected by merging into a single line.

The basic iteration in vertical and horizontal directions is very similar to the previous algorithm, but there are no specific problem cases like the vertical gaps mentioned for this algorithm. The algorithm is optimized for Western newspapers, and especially the tokenization into words, numbers and other primitives is specific to the Western writing system. The idea of using a separate layer for each text rotation is very useful, as it allows the main algorithm to concentrate only on the basic left-to-right, up-to-down case.

Both of these methods use a very geometrical approach, as they separate

text based on thresholds of whitespace. There is no significant consideration on why the methods work and what exactly is a block of text. The methods rely mostly on the proximity of text fragments, but the human perceptive system also utilizes various other cues as noted before.

2.4.2 Recursive XY-cut

Recursive XY-cut is a basic segmentation technique based on recursively cutting the document page into smaller rectangular areas [16]. The original algorithm decides the cuts based on document pixels, but there are multiple variants. A particularly interesting one is a variant that bases the cuts on projecting the bounding boxes of connected pixel components to the sides of the page [8].

The bounding box variant is based on the observation that a human reader sees the document page as resembling text even if the characters are replaced with their bounding boxes. For this reason there is no need to consider individual pixels of a document image when computing splits, which makes the computation significantly more efficient.

The basic algorithm uses a specific threshold for determining which gaps are large enough to cut at, and when to stop cutting. The end result is a partitioning of the document page into a set of rectangles, each containing content separated from others with a gap exceeding this threshold. A simplified example is shown in figure 2.4

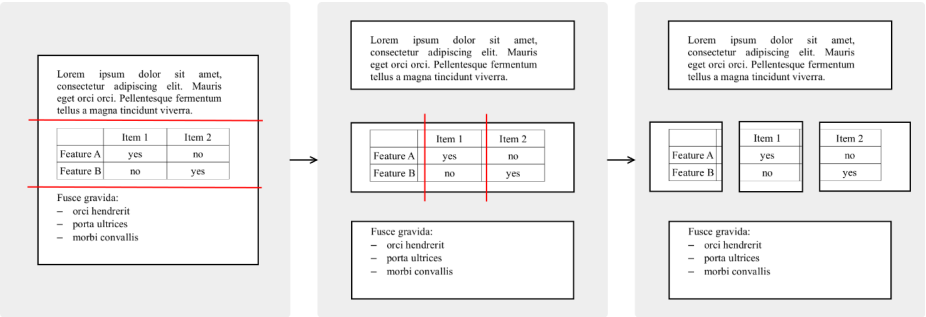


Figure 2.4: A simplified example of a recursive XY-cut algorithm. The document image is recursively cut into smaller parts until no more valid cuts can be made.

The recursive XY-cut works only on document pages that have a Manhattan layout. In a Manhattan layout, the text and graphics and other details can be separated by horizontal and vertical line segments [18]. The exact

definition for Manhattan layout varies, for example it can be understood to require or not to require all blocks of text to be roughly rectangular.

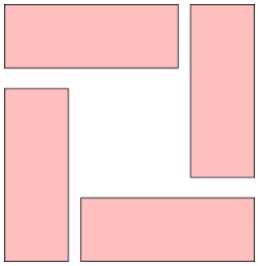


Figure 2.5: An unsolvable case for a recursive XY-cut algorithm. The blocks cannot be separated by straight cuts from page side to side.

The definition for a Manhattan layout used in this thesis is a loose one. If all the components on a document page can be separated by recursively cutting the page horizontally or vertically, the page is said to have a Manhattan layout. The exact shape of the components can be arbitrary. The case shown in figure 2.5 is non-Manhattan by this definition.

2.4.3 Detecting lists and tables

The layout of tables can be very diverse, so detecting them is a complicated problem. Many methods for detecting various kinds of tables have been devised. They generally start from OCR or image data and partly do the same processing that the block reconstruction algorithms do. Combining the table detection algorithms and detecting other components to the same system would require heavy modification of the algorithms, which are generally designed for only that one task. Instead of using the complete table detection systems, various ideas from them are used in the implemented system.

Lists have a simpler structure. They are generally one-dimensional, going from top to bottom with increasing numbering or just bullet points. Differing levels of indentation at different list levels may make parsing them somewhat complicated, but they still follow a relatively simple pattern. Some of the document analysis systems analyzed by Mao et al. [15] detect lists as one logical structure among others. A list interpreter is implemented in chapter 5, Implementation, in a similar manner, along with a table interpreter.

2.5 Existing implementations

Extracting logical structure from PDF documents is an important practical problem, and there are many other implementations that do this with varying degrees of success. These implementations include components of PDF readers and other systems that recognize structure of documents in general.

Most PDF reader applications implement some level of block reconstruction to make selection and searching of text possible. The Poppler library [22], used by many open source applications such as pdftotext and Evince, uses an algorithm that starts from individual characters and creates blocks of text. It operates in a similar manner to the algorithm described by Kieninger [10], which was considered above in the block reconstruction section. The exact algorithm and ideas used are however not described or referenced in the Poppler source code or documentation.

The de facto standard implementation for PDF rendering, Acrobat Reader, also implements text selection and search. The details behind the approach used are not available due to the closed source nature of the application, but its behavior can be compared to others. A qualitative comparison between the implemented system and Evince and Acrobat Reader is done in chapter 6, Evaluation.

A system for converting PDF documents into structured XML format was presented by Déjean and Meunier [5]. The system starts from text content extracted from PDF documents and processes it into words and lines using heuristics based on distance between characters and their geometrical positions. These heuristics are mentioned to be similar to ones used by Xpdf, from which the Poppler library originates. After preprocessing, the text is processed into paragraphs using an XY-cut approach. Desired logical structure is detected using entries from the document's table of contents as the starting points of clustering.

The PDF conversion system is implemented as separate modules which handle their own task, instead of incorporating everything into a single large model. This approach makes the system easier to understand, and ideas from it can be reused more easily as they are not tightly tied into the whole system. The system is reported to work well in the presented two use cases, which have specific document classes.

A document logical restructuring system called Dolores [2] starts from the method for finding blocks of text in PDF documents [3] that was discussed in the block reconstruction section. The Dolores system is implemented as a learning system that can be taught to recognize logical layout of a specific document class using a graphical user interface. The system relies on the

results from the physical layout analysis, and concentrates on logical labeling using a neural network. The main benefits of this approach are claimed to be high accuracy and fast interactive training for each new class of documents.

2.6 Other research in the field

Document structure recognition is a problem that has been studied widely, especially as automated understanding of documents provides great benefits in their handling and usage. A significant amount of research has been put into optical character recognition (OCR) and other visual analysis methods.

Optical character recognition is a widely studied problem. The problem has been split into multiple subproblems that can be studied, designed and improved separately. The different steps usually include document image preparation, finding appropriate features and recognizing the components of the document. These are further divided into smaller problems. [17]

The process from document image to recognizing characters and blocks of text has been successfully divided into subproblems, but the modularization of the steps after this has had less attention. The methods that find higher level structures like lists or tables or more complex layouts usually describe their actions as a single process, and leave the modularization into subproblems implicit.

When there is no clear modularization, the parts of the processes cannot be easily extracted for use elsewhere as they are too deeply tied into the complete system. They also are hard to improve independently. Most of the methods solve a very specific problem, so the research cannot be fully utilized in solving other problems when the modularization is insufficient.

Methods that have a clear modularization do exist. An experimental analysis environment for scanned documents was introduced by Rogers et al. [12]. The base problem definition is very similar to the formal definition presented in chapter 1, Introduction. A clear modularization was also introduced in a system for converting PDF documents into a structured XML format [5], which was discussed in the previous section. These modularizations are similar to the division done in chapter 5, Implementation.

Chapter 3

Environment

The environment of a system supports providing the desired functionality, but also imposes various environment specific restrictions. This chapter provides an overview of the environment in which the structural reconstruction system is implemented.

3.1 Documill Publishor

Documill Publishor [20] is a server-side application specializing in document transformations. It is capable of processing and rendering documents of various different formats, including Microsoft Office formats and PDF. The software supports various advanced rendering features, such as handling text separately from any non-textual content.

The structure reconstructor system devised in this thesis is developed to be a component of Documill Publishor. Earlier components in Publishor parse the PDF documents and provide raw text fragments from the document to the structure reconstructor, which is then responsible for finding the logical structure of the text.

The structure reconstructor component creates the structured document presentation and gives the results back to Publishor. Other components then write the structured document into the desired output format, like HTML.

Documill Publishor is implemented with the Java programming language. The structure reconstructor component and the algorithms described in this thesis are also implemented with the same language.

3.2 Input documents

The input documents of the system are of the Portable Document Format, which was described in the previous chapter, Background.

The documents given to the structure reconstruction system are not limited to any specific document type. In other words, the system must be capable of processing the textual content of any PDF document given to it with reasonable accuracy. This effectively means that the set of input documents is all the PDF documents in the world.

Being able to find the structure perfectly in all PDF documents is an overly ambitious goal, so the expectations must be lowered to fit inside a reasonable thesis work scope. A significant number of PDF documents are simple text documents with one or two columns of text, with a small number of tables and lists. This is the most significant class of input documents for the structure reconstruction system. Consequently, the goal of this thesis is that the documents of this class are processed well.

Other classes of documents such as newspapers are much more complex in layout. When processing these documents, the main goal is to find the paragraphs of text. Tables especially can be very complex, so finding the main body of tabular content is considered sufficient.

3.3 Output format

The main output format for the text content logical reconstructor system is text positioned with the combination of HTML and CSS. There are many different ways in which a text document can be presented in HTML output while preserving the original presentation of the document.

The typographical features of HTML are much simpler than they are in PDF. With custom web fonts, the correct for and layout for the character glyphs can be achieved on modern web browsers with reasonable accuracy. The positioning of individual characters cannot be controlled as precisely as in PDF, at least without wrapping every single character in an HTML element, but usually such differences are insignificant and hardly noticeable.

HTML supports paragraphs, lists and tables, so all these three reconstructed types can be written as native HTML structures. This means that the resulting HTML output will have a logical structure, whereas the original PDF did not.

HTML as an output format does not add any significant constraints to the structure reconstructor system. The combination of HTML and CSS

can be used to express any digital document as they are general purpose languages.

Chapter 4

Methods

The problem domain of reconstructing structure in PDF documents was studied in the previous chapters. Various methods, prior research and existing implementations were examined, and human perception was chosen as the starting point for finding structure independent of a specific document class.

The chosen methods will be described in this chapter. The methods are based on details of human perception, and are inspired by the previously studied research. Modularization into block reconstruction, logical segmentation and component interpretation and labeling follows what was presented in chapter 1, Introduction.

The general idea behind the methods will be explained, and the selected methods will be compared to the methods introduced as prior research. Implementation of the methods described in this chapter and the applied algorithms are considered more thoroughly in the next chapter, Implementation.

4.1 Modeling the problem

There are many different ways to describe the structure of a document. An ordinary novel could be modeled as a list of chapters, each containing headings and paragraphs, each of which contain one or more lines of text, which contain individual characters. A newspaper has a significantly more complex layout, which could be modeled as a much deeper tree structure.

Both of these models are specific to their own class of documents. While they are very powerful in describing their own type of documents, they are not useful when applied to other kinds of documents. A more generic model is required for the general documents.

4.1.1 The model

The model used in this thesis is based on blocks of text and grouping them into larger groups forming logical components. The larger groups considered here include lists, paragraphs and tables.

The blocks of text are groups of characters that are recognized as belonging together. These are modeled after the way a human reader groups characters together before recognizing their semantic meaning, as discussed in chapter 2, Background. Knowing which characters are in which blocks gives the physical structure of the document.

These blocks of text can be grouped together to find the logical structure of the document. These groups form the logical page components. In the implementation the recognized logical page components are paragraphs, lists and tables.

This model essentially views all content as part of a single layer of layout. Document layouts usually have multiple layers of logical structure, for example chapters that contain sections that contain paragraphs. However, the chosen simple model is sufficient for solving the research questions of this thesis. Recognizing further structure is considered in chapter 7, Discussion.

4.2 What information to use and not to use

The textual information on a document page can be divided into three distinct classes. The most general class is the visual, geometrical information of glyphs located on the page. Processing this information can be done with general purpose methods not limited to any class of documents.

The next class of information is the actual textual content. For example, when recognizing a table, the information whether a text line contains only numbers or also characters and punctuation can be valuable. This class of information is not language-specific, but different writing systems like Latin and Chinese text are significantly different. In a general purpose system, this information should be used to support a decision, but not to base decisions on.

Many document analysis methods utilize language and semantic information to aid processing. For example, understanding a sentence structure helps differentiating between interpretations. Understanding that “price” and numerical values near it are usually related also falls under this category. This class of information is highly specific to a single domain, for example a language or a class of documents.

The general purpose block reconstruction and logical segmentation algo-

rithms presented here utilize only visual information. The component interpreters of the segmentation algorithm may use other information if necessary. For example, recognizing bulleted lists requires understanding if a character qualifies as a bullet point. Bullet points and other list labels are generally written as text, so there is no need to use any non-textual information.

Semantic information is not utilized in the algorithms presented in this thesis. However, using this information could improve results in specific cases. This and other details such as using non-textual information is considered in chapter 7, Discussion.

The documents written in the Portable Document Format contain a lot of artificial formatting information that helps achieving the desired visual result. As noted in chapter 2, Background, this information is not very useful in recognizing the document's logical structure, and not even spaces in text writing operations can be trusted. The methods described in this chapter use text fragments written by the PDF text operators, and the fragments are additionally split further to remove any whitespace characters. This is effectively the same as using individual characters and their visual and textual information.

The fonts used in the input documents are known, and they contain additional information that is usually very accurate and useful. A very valuable piece of information is the glyph baseline position, which makes it simpler to estimate whether two characters are on the same line or not. Without the baseline information this decision would be significantly more difficult, since the visual vertical placement of different glyphs varies a lot. Especially punctuation glyphs like ' and . can be vertically far from each other. The availability of accurate font information in PDF documents is the most significant advantage compared to recognizing text with optical character recognition methods from scanned documents.

4.3 Block reconstruction algorithm

Before finding the logical layout on a page, the text fragments need to be grouped into blocks of text. A block of text contains the text fragments that are related to each other in some sense. The block may be a paragraph, a table cell, or some other perceived unit of text. The block reconstruction algorithm aims at grouping the fragments like a human reader might do before recognizing further structure, especially concentrating on proximity and similarity.

The main algorithm is based on Kruskal's minimum spanning tree algorithm. A minimum spanning tree will be constructed based on the fragments'

bounding boxes, sizes and their distance from each other. The edges that are too long compared to the size of the fragments they connect are discarded, and the result will be a set of connected graph components as illustrated in figure 4.1. These criteria effectively enforce the Gestalt principles of proximity and similarity discussed in chapter 2, Background.

Figure 4.1: A forest of minimum spanning trees over bounding boxes of text fragments. The black text fragment bounding boxes are treated as nodes, and the red edges are based on their pairwise distances.

The connected components found this way may contain one or multiple blocks, so they need to be split further. The components contain only complete blocks, so each component can be processed individually independent of other blocks. The text size inside a component is consistent, so it does not need to be taken into account.

The splitting of the connected components is done by finding holes of whitespace inside them. If the hole is large enough, the component is split around it. The exact heuristics used for determining the split are described in chapter 5, Implementation.

There are multiple details in the block reconstruction algorithm that depend on the input format. Measuring the text size can be problematic, and finding the spaces in a text line is non-trivial. The way these can be done in the PDF case will be discussed in the implementation in chapter 5, Implementation.

The block detection algorithm is in essence a connected component algorithm, but viewing it as a minimum spanning tree problem gives a useful point of view. From this point of view, the definition of the graph can be made implicit and incorporated into the algorithm, eliminating the need of creating an explicit graph.

4.4 Logical segmentation and labeling

Logical segmentation is the process of segmenting content on a page into valid logical components. The goal is to know which components exist on the page, what type they are and what are their contents.

The segmentation of the page is based on the same idea as the recursive XY-cut algorithm. This variant operates on the bounding boxes of the blocks of text detected by the previous block reconstruction algorithm. In the projection, only the completely empty parts of the projection are considered, instead of allowing small overlapping.

The segmentation algorithm utilizes component interpreters that are able to recognize content as meaningful from their own point of view. Consequently, each iteration finds only one gap around which the current region is split.

The segmentation algorithm utilizes only geometrical information. The component interpreters may use any information relevant to recognizing the class of components they recognize.

4.4.1 Component interpreters

The logical segmentation algorithm utilizes specialized interpreter modules to aid in segmenting the page into logical components. An interpreter specializes in recognizing its own class of components, such as a table or a list, much like the indicator functions in the problem formalization in chapter 1, Introduction. When segmenting a document page into smaller parts, the segmentation algorithm will query the interpreters with candidate groups of text blocks.

The interpreters have two operations, known as glance and interpret. The glance operation is light-weight and tells if the component given to it looks like it could be classified as what the interpreter recognizes. For example, a table interpreter checks if the given component seems to have a tabular structure in it. The glance operation is allowed to give false positives.

The interpret operation is heavy-weight and does the actual recognition of structure. This operation reconstructs the structure and tells if it succeeded or failed in it. The recognized component does not need to use all the blocks in the candidate component.

The logical components relevant to this thesis include lists, paragraphs and tables. The exact way their interpreters are implemented is covered in chapter 5, Implementation.

4.4.2 Segmentation

In the segmentation, the bounding boxes of the blocks detected by the block reconstruction algorithm are projected to the X and Y axes of the document page. This results in gaps in places where there are no blocks as illustrated in figure 4.2.

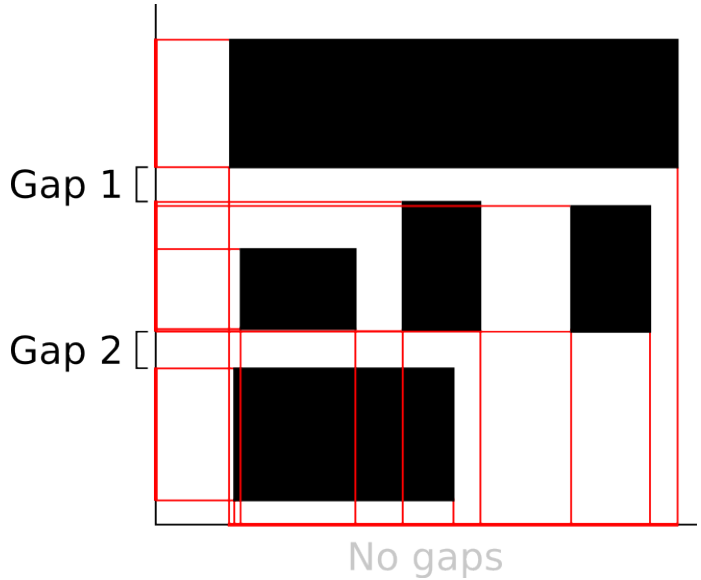


Figure 4.2: Projecting bounding boxes of text blocks onto the X and Y axes. Two gaps are found on the Y axis, and none on the X axis.

Each of these gaps cuts the document into two parts when considered separately from each other. One of the cuts should be chosen to be able to continue with the segmentation. The resulting two parts should each contain only complete logical components, and splitting a component into two parts should be avoided. The goal of the segmentation is finding the components, so they are naturally not yet known and various heuristical methods must be applied instead.

To select the optimal gap to cut at, the parts created by splitting around the gaps are offered to each interpreter using the glance operation. Since glance is allowed to give false positives, the split parts are not yet interpreted as logical components, but they are candidates for that. For each cut, the interpreters identify zero, one or two possible candidates that can be logical components. This is illustrated in the figure 4.3.

As there will likely be multiple gaps with the same number of possible candidate logical components, a secondary heuristic is required to select between these. The component interpreters recognize less candidates at the

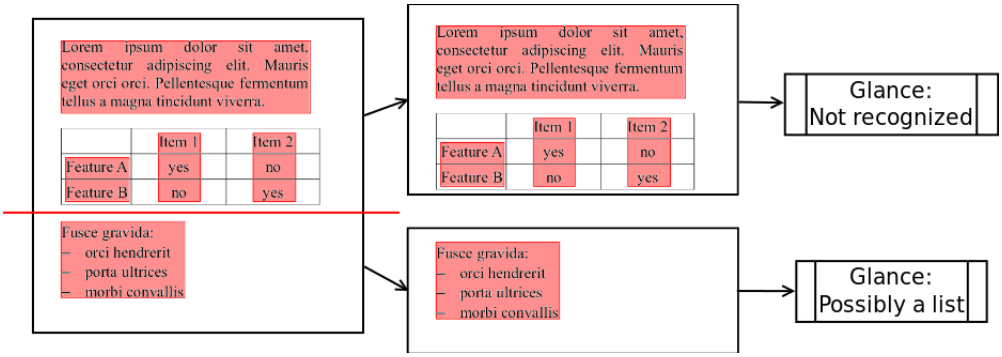


Figure 4.3: Cutting a section of a page into two candidate components. The selected cut leads to one candidate component being recognized as a possible list by list interpreter's glance operation.

start of the segmentation, so the secondary heuristic is dominant at that point.

To select between gaps that would lead to equally good sets of candidates, the width or height of the gap is taken into account. The larger a gap, the more likely it is to separate content appropriately. If there are multiple gaps with a similar width or height, the first or last of them should be selected to preserve good continuation. If there is a very large gap, it can be selected even if it would result in less valid candidates for logical components.

After selecting the gap used for splitting, the current set of bounding boxes is split around the gap. If a half is recognized by some interpreter's glance operation, the interpreter is requested to interpret it with the interpret operation.

If an interpreter finds a component such as a list or a table inside the section of the document given to it, the found component is marked and the remaining parts of the current section (if any) are segmented recursively further.

If the half is not recognized by any interpreter or all the interpreters that recognize it fail to interpret it, the segmentation is recursively continued on that part. The overall process is illustrated in the figure 4.4.

4.5 Analysis

The presented block reconstruction algorithm first joins the text fragments into blocks of text, starting from local features and progressing towards the global structure. The found blocks are then split if they contain too large gaps between fragments. This approach builds mainly on the simple Gestalt

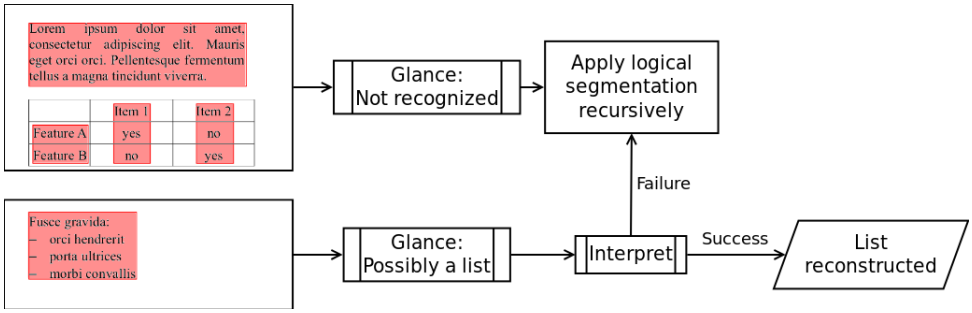


Figure 4.4: The interpret operation applied to the result in figure 4.3. If it succeeds, a list is successfully reconstructed. If it fails, the candidate component is recursively split further.

principles, especially the principles of proximity and similarity.

The reviewed prior research on human perception indicated that the visual processing order is neither global-to-local or local-to-global, instead being more opportunistic. The splitting heuristic in the block reconstruction algorithm makes it avoid being strictly local-to-global, and is opportunistic in some sense. The logical segmentation algorithm, on the other hand, is a global-to-local method that utilizes component interpreters that opportunistically detect their own types of page components.

The logical segmentation algorithm shares the common problem of XY-cut algorithms and can not be expected to work perfectly on non-Manhattan document layouts. However, the opportunistic behavior of the interpreters might help solve some non-Manhattan cases, as the interpreters are allowed to recognize only part of the candidate component and return the rest for further processing.

The actual implementation of the presented methods will be presented in the next chapter, Implementation. This includes the descriptions of the list and table interpreters, which were omitted from this chapter. Extending the block reconstruction algorithm to support rotation and other future work is discussed in chapter 7, Discussion.

Chapter 5

Implementation

The document logical structure reconstruction process is divided into block reconstruction and logical segmentation, as described in the previous chapter and shown in figure 1.2. The blocks found by block reconstruction will be processed further using logical segmentation and component interpreters, which guide the segmentation. The implementation of each of these will be described in this chapter.

This chapter covers the implementation details of the methods and algorithms described in the previous chapter, Methods. The benefits and drawbacks of the selected implementation are considered at the end of this chapter. This will also include considerations on how the environment described in chapter 3, Environment, affected the implementation.

The actual data structures used in the implementation are not described in detail. The same ideas and algorithms can be implemented using any of the various kinds of list and set structures, and a simple list supporting random access is usually enough. Optimization and more advanced data structures are considered in chapter 7, Discussion.

5.1 Selection of parameters

The implemented methods use heuristics based on numerical constant parameters. The parameters describe thresholds for various rules, for example how far a text fragment is allowed to be from another to be deemed close. The parameters used are listed in the table 5.1. The proximity, width and height parameters are measured as multiples of font size.

The selection of the constants was done by intuition and they were adjusted when necessary. This means the resulting constants are not necessarily optimal, but they have been observed to work on a large number of docu-

Parameter	Value	Description
d_{filter}	3.00	Threshold for filtering out too long edges in preprocessing
d_{fh}	1.01	Horizontal proximity threshold of fragments
d_{fv}	2.01	Vertical proximity threshold of fragments
d_{base}	0.1	Maximum vertical offset of aligned baselines
s_{font}	1.25	Similarity threshold of font sizes
s_{fontb}	2.11	Similarity threshold of font sizes, baselines aligned
s_{grid}	0.01	Similarity threshold of grid line coordinates
w_{s1}	1.00	Minimum width of a splitting rectangle, case 1
h_{s1}	3.00	Minimum height of a splitting rectangle, case 1
w_{s2}	5.00	Minimum width of a splitting rectangle, case 2
d_{line}	0.20	Maximum baseline distance of a fragment from its line
$s_{spacing}$	0.10	Similarity threshold of line spacings
c_{gap1}	3.00	Relative size above which a gap is always selected
c_{gap2}	0.33	Relative size above which to select a more promising gap
c_{gapa}	0.25	Size difference for selecting a similar gap above current one
c_{gapb}	0.25	Size difference for selecting a similar gap below current one
t_{lwf}	4	Threshold for filtering out lines too wide relative to median width
d_{align}	1.00	Table column alignment threshold
d_{jump}	2.00	Threshold of too large jump in gap size relative to previous
t_{used}	0.50	Minimum ratio of candidate cells used in table
t_{cmax}	30	Maximum average number of characters per table cell line
t_{clong}	15	Long table line threshold
t_{fratio}	0.75	Minimum fill ratio of table cells
$t_{overlap}$	0.50	Table bounding box overlap threshold

Table 5.1: Table of parameters used in the heuristics in the implementation of the logical reconstruction system.

ments.

5.2 Block reconstruction

The block reconstruction algorithm implementation follows the description in the previous chapter. The fragments are manipulated in a single list data structure, which is split into multiple smaller parts. The algorithm first finds the connected components of the implicit graph described in the previous chapter, and then splits the connected components into smaller parts if needed.

5.2.1 Finding the connected components

The block reconstruction starts by finding the sets of fragments that are close to each other. This is done by considering all pairings of fragments as edges of a graph. The length (weight) of an edge is the minimum Manhattan distance (the ℓ_1 norm) from any point from one fragment's bounding box to any point in the other's bounding box.

The Manhattan distance between the bounding boxes can be calculated by projecting the edges of the bounding boxes to the X and Y axes and measuring the lengths of the projections. If the projections overlap on either one or both axes, the distance in that dimension is 0. The vertical distance is the distance on the Y axis, and horizontal on the X axis. The Manhattan distance is then the sum of these two distances. The projections are illustrated in figure 5.1.

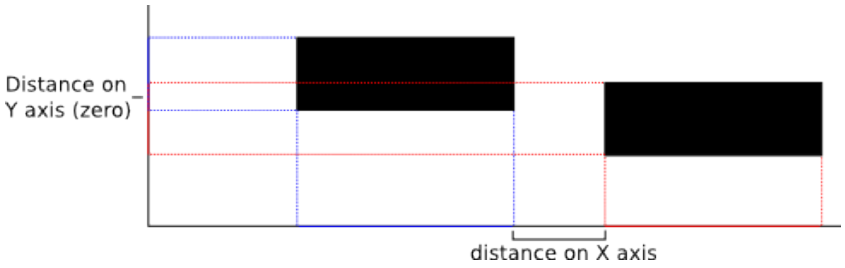


Figure 5.1: Measuring the Manhattan distance between two bounding boxes using projections to axes.

Finding the minimum spanning tree starts by listing all the edges. The edges that are longer than d_{filter} times the font size of either fragment are considered too long, and they are omitted already at this point to reduce the number of edges to process. The fragments are also required to be either horizontally or vertically aligned, so the edges with both non-zero vertical and horizontal distances are also discarded. These edges are then sorted to increasing length, so that shorter edges are considered before longer ones.

The sorted list of edges is then processed one edge at a time to determine whether the edge qualifies as part of the minimum spanning tree. If the fragments connected by the edge are already in the same set, the edge is discarded. If the edge's vertical length is zero and the edge's horizontal length is over d_{fragh} times the minimum font size of the two fragments, the edge is discarded as too long. If the horizontal length is zero, the same happens when the vertical length is over d_{fragv} times the minimum font size. These parameters are illustrated in figure 5.2.

The baselines of the fragments connected by the edge are considered to

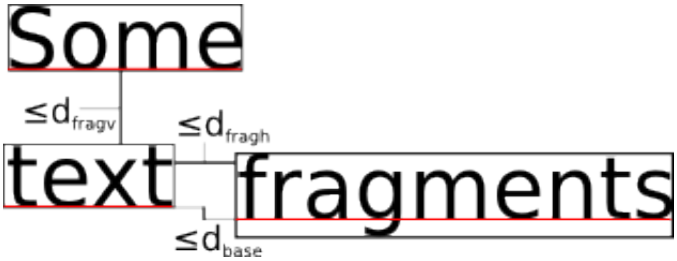


Figure 5.2: Distance thresholds between fragments and their baselines.

be aligned if the baseline Y coordinates differ by at most d_{base} times the font size of either fragment. The larger of the font sizes of the two fragments can be at most s_{fontb} times the smaller one if the baselines are aligned, and s_{font} otherwise. Additionally, if the fragments are left and right of each other, they are required to have aligned baselines, as defined above.

The fragments connected by the edges that pass all these requirements are then combined into the same set, along with all the fragments that were in the same sets as these two. This is done efficiently by using a disjoint-set data structure with a union-find algorithm [23].

After all the edges are processed, each fragment has been added to some set. These sets are the connected components of fragments close to each other. The sets of fragments will often also have a uniform font size, which is a useful property for the later processing steps.

This algorithm is essentially Kruskal's minimum spanning tree algorithm. The graph is implied by the requirements for the edges included in this algorithm. Using Kruskal's algorithm would result in a set of trees whose nodes contain the same fragments as the sets of this algorithm.

5.2.2 Splitting the connected components

The found connected components are analyzed further by creating a grid based on the text fragments' bounding box coordinates. The left and right X coordinates and top and bottom Y coordinates of each fragment are collected and sorted. In both dimensions, differences less than s_{grid} times the font size are considered insignificant and are removed. The grid is illustrated in figure 5.3.

These coordinates give a grid with at most twice as many columns and rows as there are text fragments. The fragments are inserted to this grid by finding the right cells with binary search and marking them as filled.

The grid is inspected further by finding the largest rectangles [24] of non-filled cells. If the width of the rectangle is at least w_{s1} times the font size

and the height is h_{s1} times the font size or larger, the rectangle qualifies as a potential splitter. If the width of the rectangle is more than w_{s2} times the font size, it also qualifies as a potential splitter.

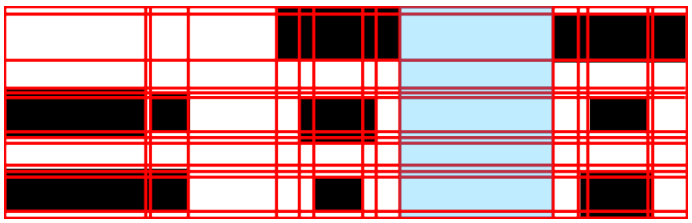


Figure 5.3: A grid based on the sides of bounding boxes of text fragments. The largest rectangle of non-filled cells is highlighted in light blue.

These splitting rectangles are considered in the order of decreasing size. If either left or right side of a rectangle has no fragments, it does not qualify as a splitter and is discarded. The first rectangle to have fragments both left and right is used to split the connected component. Usually there is no splitting rectangle, and the connected component is simply left as it was.

The actual splitting is done by dividing the connected component into four sets around the rectangle. The fragments above and below the rectangle form their own sets, and the remaining fragments left and right form the last two sets. These sets of fragments are individually processed again by starting the block reconstruction on each set separately.

5.2.3 Finding the text lines

After splitting the connected components in the previous step, they are assumed to have only a single column of text. The text lines are also assumed to be horizontal. Due to the way the connected components are found, the components are known to contain fragments that are close to each other in font size.

To find the text lines, the average font size of the fragments is calculated. The PDF format supplies baseline information for each font glyph, so this can be used to estimate the location of the line's vertical center. All fragments with baseline Y coordinates differing by at most d_{line} times the average font size from a center are grouped together to form a line.

Grouping is implemented by sorting the fragments by baseline Y coordinate, starting at the first line and marking it as the line center. Fragments are added to the line and the center Y coordinate is updated until the next fragment is d_{line} times the font size or more away from the line. The process

is repeated with the next fragment until all fragments have been processed. The line grouping is illustrated in figure 5.4.

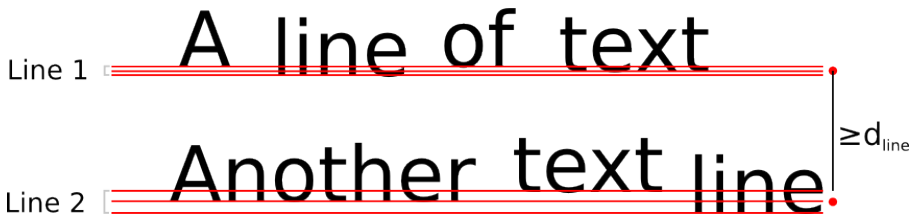


Figure 5.4: Forming lines from individual text fragments by grouping baselines based on their distances from each other.

5.2.4 Finding the text blocks

The connected component has been split into lines, but it may still have several different line spacings. The vertical distances between consecutive lines are calculated, and divided into groups by considering distances differing by at most $s_{spacing}$ times the average font size to be equal. This grouping of lengths is done in the same way as grouping text fragments into lines, with the distances being used in place of Y coordinates.

This produces a set of line spacings, with usually from one to three distinct values. The lines are then joined into blocks by iterating from the smallest line spacing between two lines to the largest.

In each iteration lines are joined into a block if neither of the lines has been joined to a block already, or if the space between the lines matches the line spacing currently being considered. After repeating this for all the spaces between lines and all the line spacings, the text blocks have been created.

5.2.5 Additional processing

Some documents place multiple text fragments closely on top of each other, creating a bold or shadow effect. These cases are solved by finding overlapping text lines, splitting them into individual characters and removing overlapping characters at each occurrence of an overlap. The character that was extracted last from the PDF file is kept, and all the others that overlap it are removed.

Subscripts and superscripts are handled by attaching them to appropriate lines after the blocks have been found. This approach works when there are few of superscripts and subscripts, but larger amounts affect the block detection and produce worse results.

A single block may have multiple paragraphs in it, as only the line spacings were considered when creating the blocks. Paragraphs may be denoted by varying indentation levels. If only a few of the lines are indented, these are assumed to denote paragraph starts and such blocks are split into smaller paragraphs.

5.3 Segmentation algorithm

The logical segmentation algorithm that finds the logical components on the document page is implemented as a recursive segmentation algorithm and interpreters for each logical component type (paragraph, list, table). The implementation follows the description in chapter 4, Methods.

The segmentation algorithm starts by considering all the blocks of text found by the block reconstruction algorithm as a single candidate component. The component will be offered to all component interpreters using the glance operation. If at least one interpreter claims to recognize it, it is asked to interpret it with the interpret operation. If an interpreter succeeds in interpreting the component, the component is marked as the corresponding type.

If none of the interpreters recognizes the candidate component, it will be split. This is done by projecting all of the blocks inside the component onto the X and Y axes to find the gaps between the blocks. To do this, the range from component bounding box minimum X to its maximum X is first calculated. Next, the X coordinate ranges of the blocks inside the component are removed from this range. The same is done for the Y ranges, and the result is the horizontal and vertical gaps between the blocks as was illustrated in figure 4.2.

Each of these gaps splits the candidate component into two parts, which are two new candidate components. The two parts are offered to the interpreters using the glance operation, and this is repeated for all the possible gaps.

Selecting the optimal gap follows the principle given in chapter 4, Methods. The gap with the most recognized components (as determined with glance operation) is favored, and the top or bottom gap among similar gaps is selected.

The selection starts by considering the gaps from top to down, and left to right. The first gap is marked as the currently best result. If the next gap is more than c_{gap1} times larger than the previous one, it will be marked as the new best result. If the next gap is at least c_{gap2} times the previous one and it has more recognized components, it will be marked as the best. If the

number of recognized components is the same but the gap is larger, it will be marked.

If there are multiple consecutive similar gaps, the first or last of them should be selected to preserve good continuation. To select the topmost or bottommost between similar gaps, the similarity of consecutive gaps is analyzed. If selecting either of the gaps results in a recognized component right of or below the gap and non-recognized on the other side, the upper or lefter gap will be selected if the other gap is at most c_{gapa} larger. Similarly, if there is a recognized component above or left and none on the other side, the lower or righter gap will be selected if it is at most c_{gapb} smaller.

A small optimization is done if there exists a pair of horizontal and vertical gaps so that only the top left and bottom right fourths have content, or the top right and bottom left fourths. In this case either of the two gaps is selected and splitting is done accordingly.

After the best gap has been chosen, the component will be split. The resulting new candidate components will be processed recursively further, starting from offering them to the interpreters with the glance and interpret operations as explained above.

5.4 Table interpreter

There are many variants of tables, some simpler and some that have a very complex structure. The table interpreter concentrates on the simple case where the table columns are aligned to left, center or right.

The interpreted table cells are allowed to contain either individual lines or blocks of text, but not both.

5.4.1 Glance step

The glance step of the table component interpreter starts by collecting all the lines in the given component into a single list data structure and finding the median width. Next, all the lines with width of over t_{lwf} times the median are filtered out so that single long lines that might affect the following projection are removed.

The remaining lines are fit into a table by projecting them to the X and Y axes and using the gaps to find a simplified interpretation of cells. If this table has less than two rows or columns, the glance step ends with no recognition. Otherwise, the candidate component is reported as possibly containing a table and being suitable for the interpret step.

5.4.2 Interpret step

The interpret step attempts to interpret the candidate component as either one of the supported types, a table of blocks or a table of lines. This is done using the same algorithm for both types, but having the basic unit be a block or a line depending on the case.

5.4.2.1 Initialization

The candidate cells of the table are found using the basic recursive XY-cut algorithm on the bounding boxes of the lines or blocks and cutting until there is no non-zero gap left. These cuts partition the overall bounding box of all the blocks or lines into rectangle cells. Additionally, for each cell, the closest neighbors above, below, to the right and to the left are then determined by finding the closest cell in the selected direction. If there are multiple possible choices, the one with the longest side facing the current cell is chosen.

5.4.2.2 Expansion

After finding the neighbor relation, the centermost cell is selected as the seed of the table. The table is iteratively expanded in vertical and horizontal directions starting from this center cell. When considering horizontal expansion, the table is first expanded to the left and then to the right. This is repeated until either no more expansion can be done or the table was expanded more than five times. This means six expansions can occur if in the last iteration the table was expanded to both left and right. The vertical expansion works in the same way but to up and down instead of left and right.

The expansion to a given direction starts by considering the closest candidate cells in that direction. To find the cells above the current table, the above area is formed as a rectangle using the width of the current table's bounding box, the position of the upper side of the bounding box, and the height of the page. All the candidate cells overlapping this rectangle are considered to be above the current table.

The found cells above the table are projected to the Y axis, which gives the gaps between the cells. The cells below the lowest gap are then the closest cells above the current table, and are considered for addition to the table. Finding the expansion in the other directions works in the same way, with the rules rotated accordingly.

When the new set of candidate cells has been found, it is determined whether they are a good addition to the current table. If the combined bounding box of the table and the new cells overlaps some cells that are not

already in the table or are not being added, the table is not expanded to that direction.

To find the row and column structure of the candidate table expansion, the candidate cells' contents are projected to the X and Y axes to find out the gaps between the cells. This gives the candidate rows and columns. If there are more candidate cells than rows times columns, the cells that span the whole width or height are removed and the projection is done again. The cells, including the spanning cells, are then added to the candidate expansion based on the found rows and columns.

The candidate table expansion is then merged to the current table. If the bounding box of the combined table overlaps some other candidate cells not in the table, merging is aborted and the table expansion is rejected.

The rows or columns of the expansion must match the rows or columns of the current table. This is checked by attempting to intersect the lists of gaps between the rows or columns. A gap can be intersected with another if the coordinate ranges overlap. The lists are intersected by moving over the two sorted lists and considering pairs of gaps. If the two gaps can be intersected, their intersection is added to the intersected gap list. If they cannot be intersected, the one that is left or above the other is added to the intersected list and the other one is considered for the next intersection. If this happens more than once, the expansion is aborted.

When expanding up or down, the new cells must fit the columns of the current table. The columns can be either aligned to left, right, or centered. If the candidate table expansion does not fit the current one, it is rejected. The X-wise coordinates of the left, center or right side of the cell contents are allowed to differ at most d_{align} times the font size from each other to be considered properly aligned. If the expansion candidate contains spanning cells, the column checking is skipped for the affected columns. If the expansion candidate contains multiple columns where the current table contains only one, the new ones replace the affected column in the current table.

The gaps between columns and rows of the table are next analyzed. The lengths of the gaps in a given direction are sorted, and the second half of it is considered. If there is a gap that is more than d_{jump} times the previous one, the jump in size is considered too large and the average of the two gaps is used as a margin for trimming the table. The longest sequence of gaps that are less than the margin wide or high is then used, and the columns or rows not next to the gaps are trimmed away.

5.4.2.3 Validation

After the iterative expansion has been done, the resulting table is analyzed. If the resulting candidate table has less than two rows or columns, it is rejected. If the candidate table uses more than t_{used} of the candidate cells, it is accepted. If it uses less than half the candidate cells and there are excess cells in three or four directions out of above, below, left and right of the table, it is rejected. If there are excess cells in two or less directions, the table is accepted.

After finding the candidate line and block tables, many further heuristics are used to determine which ones are good tables and which one should be selected. First, the found line table, if any, is examined. If the text lines in the table contain more than t_{cmax} characters on average, the table is determined not to be a good table. If the lines have more than t_{clong} characters on average and less than half of them are numbers, the table is not good either.

Next, both the found line and block table are examined. If less than t_{fratio} of the cells are filled, the table is not good. If the table has a corner cell that is filled but none of the cells on the same row or column are filled, the table is not good. If the rightmost column of the table has only a single filled cell the table is not good either. In other cases, the table is determined to be good.

If both a good line table and a good block table is found, the choice between them is done based on the total size of the tables, and whether the block table contains the line table, which can happen if there are multiple tables on the same page. If the line table's bounding box is larger in area than the block table's bounding box, the line table is always selected.

If the overlap of the two table's bounding boxes is at least $t_{overlap}$ times the size of the line table's bounding box, the line table may be contained in the block table. This is tested by determining if there's a block in the block table that is neither fully in the line table nor fully out it. If such a block does not exist, the line table fits the block table and the line table is selected. If the line table was not selected by now, the block table is determined to be the better one.

If no good table is found, the candidate component will be given back to the segmentation algorithm for further segmentation. Otherwise, the text content inside the found table is marked as a table, and the remaining text is given back to the segmentation algorithm.

5.5 List interpreter

Lists are a set of list entries, which contain list labels and list bodies. A single list entry may be written on multiple lines of text, where the first line contains the list label and the start of the list body. The list structure is illustrated in figure 5.5.

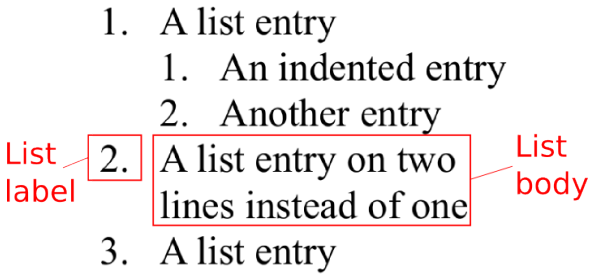


Figure 5.5: A simple indented list. The list label and body are highlighted.

The list label might be separated from the body so far that the block reconstruction algorithm fails to recognize is as part of the same line. In this case the label must be attached to the line in the list interpreter.

A list may have multiple levels indicated by larger indentation and possibly different type of labels. These can be viewed either as sublists or simply as differently indented entries. List labels have multiple types, including for example bulleted and numbered lists. A list can also span from one page to another, so the first entries on a page can already be indented. These details are taken into account in the interpret step.

5.5.1 Glance step

The glance step of list component interpretation creates a single column table out of the component given to it. This is done by projecting the lines into the Y axis and using the gaps to find the cells. If one of the cells contains more than two lines of text, the component is deemed not a list. If the list label is separated from the body, there can be two lines of text in the same cell.

If a vertical gap between the contents of consecutive cells is too large, the component is not a good list. If there are two lines in a single cell but neither looks like a list label, for example ”1)” or ”4.”, the component does not make a good list.

These heuristics filter out the candidate components that do not seem to be interpretable as lists in their current form. There may be list-like

features present, but the other contents should be removed by continuing segmentation before attempting to interpret a list in it.

5.5.2 Interpret step

The interpret step begins by creating a single column table out of the component. After the glance step it is known that each cell contains either only a single line or two lines with one that looks like a list label.

The cells of the table are processed as follows. If there is a single line in the cell, it is examined for a list label. If a list label is present, the line is marked as containing a label and part of the body. If there are two lines in a cell, they are concatenated into one line marked as containing a label and part of the body.

The lines found this way are collected and examined further. The whole component might not be a list, so each continuous subsequence of the lines is considered separately by decreasing length and increasing start point.

Each subsequence examined for whether it is a valid list or not. The first line must have a list label, and each line after it must have either non-label text or a label and other text. By grouping each non-labeled line with the preceding labeled line, the candidate list entries are received.

In each list entry, the text lines must be indented at a higher level than the leftmost list label in the list. The indentation levels are calculated by comparing the X-wise difference of the line's left side compared to the leftmost label's left side. These values are quantized by dividing with the font size, and normalized so that the values start from zero and increase one step at a time.

At each indentation level, the list labels should be consistent. For example, having list labels like "1." and "1.4." at the same level does not make a valid list. The list entries should also have consistent line spacing.

Each valid list subsequence found this way is marked as a list. The remaining lines are marked as paragraphs, as they are not lists and have only one column of content at this point.

5.5.3 Further discussion

One way to define list label format is using regular expressions. For example, the regular expression `[0-9\\.]*[0-9]+\\.` can be used to recognize list labels like "1.4." and "124.2.".

Other types of list interpreters can be created with the same principle as explained here, or a list interpreter can be extended to interpret multiple types of lists. When using one interpreter for all lists, the list label types

may conflict with each other. To solve this, each type should be considered and the one most suitable chosen.

5.6 Paragraph interpreter

The paragraph interpreter is implemented implicitly. If the other interpreters do not recognize the component and it cannot be split further, it is marked as a paragraph.

This implementation is almost identical to one that recognizes components containing only a single block as a paragraph. The differences include that the other interpreters are always selected if they are able to interpret the component, and components that cannot be split by the segmentation algorithm are marked as paragraphs.

5.7 Analysis

The implementation contains a large number of heuristics using threshold values for determining whether a certain operation or interpretation should be chosen. These were chosen and optimized manually while testing the system on various documents during development.

The table interpreter is relatively complex compared to the other parts of the system, even though it was designed to recognize only relatively simple tables. Its development was found to be difficult using simple rules and an imperative programming approach, the use of which was effected by the system environment. Adding more checks to the glance step could reduce the complexity.

The glance step of component interpreters is allowed to recognize candidate components that the interpret step may not be able to interpret. Especially the table interpreter has a very simplified glance step. This may guide the logical segmentation algorithm to split the current document area using an inoptimal cut. However, the background studies have found human perception to be opportunistic, so this might not be a problem. If the human perception makes similar decisions, it may have been taken into account implicitly in the layouts of human readable documents.

The block reconstruction algorithm compares the sizes of individual text fragments when considering joining two fragments and their groups. Comparing the average of the sizes in the current groups could be more suitable. However, this would matter only in cases where the size keeps increasing

in some special pattern. Grouping such sets of fragments could also be the appropriate interpretation, depending on the case.

The logical segmentation algorithm defaults candidate components to paragraphs if it can no longer split the candidate component and no interpreter recognizes it. Creating a explicit paragraph interpreter could be a better idea, for example recognizing components containing only a single block of text as a paragraph. This would allow treating candidates that cannot be split in some other way.

The behavior of the system will be studied in the next chapter, Evaluation. The results and the problems related to rule-based systems and parameter optimization are considered in chapter 7, Discussion.

Chapter 6

Evaluation

After the implementation of the system, it is important to test that it works correctly and to analyze how well it works. In this chapter, the structure reconstruction system will be evaluated using a set of test documents. Both a quantitative and qualitative analysis is done based on this set of documents.

Additionally, the performance of recognizing blocks of text is compared to equivalent behavior in Acrobat Reader and Evince. This qualitative analysis will be done using a small set of specifically crafted documents that represent hard cases for structural reconstruction.

These results will be compared to other similar methods for structural reconstruction, concentrating on both the reported accuracy and the properties of the analyzed systems. Further discussion on the strengths and weaknesses of the chosen approach is done in the next chapter, Discussion.

6.1 Evaluation process

The performance of the structure reconstructor was evaluated using a set of twenty documents. The set of documents was collected by searching for PDF files and suitable keywords, as this method most closely resembles how the system is used in practice.

The documents were evaluated by manually analyzing the results and counting, for each component type, how many components were detected correctly and what kind of problems there were. In cases where there were multiple valid interpretations, any of them was considered correct. Further occurrences in the same document were required to be consistent with the previous interpretation, however.

6.2 Test documents

Twenty documents with a total of 390 pages were collected. The documents contain paragraphs, lists and tables, although not all documents contain both lists and tables. The layouts of the documents are simple, but very heterogeneous.

The documents are operational reports from various student organizations and companies, ten from both groups. They vary from simple text documents to scanned documents with text from an optical character recognition system. The document pages contain mostly one or two columns of text.

Another, simpler set of documents is used for comparing the implemented system to PDF reader software. This document set is described in section 6.4, Comparison to PDF reader software.

Both sets of documents are listed in appendix A, Test documents.

6.3 Results

The evaluation results are listed in the following tables. Evaluation results for blocks of text were collected separately for blocks of a single line (table 6.2) and blocks of two or more lines (table 6.1). The results for tables are listed in table 6.3 and for lists in table 6.4.

The evaluation results tell, for each corresponding page component, how it was reconstructed and classified. Common faults include the component being split into multiple pieces or being joined with another of the same or different type. Tables may be both split and joined at the same time.

A few times a table of lines was interpreted as a table of blocks, or the other way around. If they were otherwise correct, they were marked as being the wrong type. If a list was otherwise correct but the indentation levels were interpreted wrong, these were marked as having wrong indentation. Both tables and lists could also be completely missed and were marked as such.

Paragraphs			
Correct	Split	Joined	Total
1678	271	331	2280
73.6%	11.9%	14.5%	100.0%

Table 6.1: Evaluation results for paragraphs of text.

In addition to these recognized types, there were 152 logical components that were not paragraphs, lists or tables. These include, for example, bal-

Single lines			
Correct	Split	Joined	Total
1529	19	486	2034
75.2%	0.9%	23.9%	100.0%

Table 6.2: Evaluation results for single lines of text.

Tables						
Correct	Split	Joined	Both	Wrong type	Missed	Total
18	74	5	9	2	12	120
15.0%	61.7%	4.2%	7.5%	1.7%	10.0%	100.0%

Table 6.3: Evaluation results for table interpreter.

ance sheets and tables of content. Since no interpreter recognizes them, the statistics for them were omitted.

6.4 Comparison to PDF reader software

The structure reconstruction system was compared to some freely available PDF reader and text extraction applications. The compared software include Acrobat Reader, Evince and pdftotext.

Acrobat Reader and Evince are viewer software that allow selecting text in the viewed PDF document, which requires recognition of blocks of text. Pdftotext is a command line utility for extracting blocks of text from PDF documents. Acrobat Reader supports both viewing the text and saving the document as text. These operations seem to be separate implementations and they give different results. Both Evince and pdftotext are based on the Poppler library [22], and they give essentially similar results.

The software and the system are compared on a test set of 15 small documents, which is listed in appendix A, Test documents. Each document in the test set represents some aspect that may be difficult for a reconstruction system. For example, one test document contains blocks of text that cannot be separated from each other by cutting from document side to side using the basic XY-cut algorithm. Another document contains a block of text, and two columns of text wrapping around it. The documents were created using LibreOffice Writer and Scribus.

Out of the 15 test documents, the structure reconstruction system handles 11 perfectly. Exporting raw text in Acrobat Reader works perfectly in 9 cases, but only in 3 cases with pdftotext. Acrobat Reader and Evince implement text selection in the correct reading order in 13 and 11 documents.

Lists					
Correct	Split	Joined	Wrong indentation	Missed	Total
211	24	19	7	51	312
67.6%	7.7%	6.1%	2.2%	16.3%	100.0%

Table 6.4: Evaluation results for list interpreter.

The PDF reader and text extraction software do not implement any list or table detection. As a result, lists with varying levels of indentation are flattened into a single indentation level. The structure reconstructor system handles the two documents testing this aspect as designed.

There are some indications that viewing text in Acrobat Reader relies somewhat on the internal PDF representation of text. Selecting text in the viewer often works surprisingly well, but in the raw text output the same result can be messy. In some cases determining the correct order would require advanced heuristics, and not utilizing them also in the raw text output would be illogical. This is particularly evident in the following case.

A test document (newspaper_wrapped in appendix A, Test documents) contains two columns of text that wrap around a block of text. Text selection works well in Acrobat Reader's viewer, but all other software break, including Acrobat Reader's raw text output. In the structure reconstruction system the lines in the middle block of text are joined to the lines left and right, but the lines above and below the middle block are logically grouped into blocks. The output of pdftotext joins the lines, and then outputs the remaining lines in a seemingly chaotic order. Acrobat Reader's raw text output splits the problematic lines into individual words, and then gives a result similar to pdftotext.

The structure reconstruction system finds rectangular gaps in text block candidates and splits them into above, below, left and right sections. This causes the text to be split into four pieces in a test case (text_wrap) containing text wrapping around an empty rectangle. Pdftotext gives a similar result. The raw text output of Acrobat Reader is somewhat similar, but the lines on the right are joined to the next lines on the left. Text selection in Evince and Acrobat Reader functions like in the original document. Handling this test case perfectly would require linguistic analysis to tell whether the text continues over the rectangle or if the left and right sides are separate.

One test document (reverse_text) visually contains the string "this is text", but in the PDF file it is written in reverse by abusing kerning. The structure reconstruction system handles this, and so does the save as text functionality in Acrobat Reader. Both Evince and pdftotext give "t h i s i s t ex t", and copying text from Acrobat Reader's viewer gives "th i s i s text".

Large holes inside blocks caused by bad justification (e.g. justification_gaps) are problematic for every implementation that exports blocks of text. Viewing text in Acrobat Reader or Evince works, but saving document as text in Acrobat Reader can result in a broken output. Pdftotext generally manages to maintain the correct reading order of the block contents, but the blocks themselves are split. The structure reconstructor also suffers from similar problems if the justification holes are especially large and numerous, but it handles the basic occurrences finely.

Some of the test cases are designed to expose simplifications made in the design of block reconstruction algorithms. These include a paragraph written in a wave-like pattern instead of a rectangular text alignment, having spaces between words align vertically forming "rivers of white", and some other cases. The tested systems do not have any problems with these test cases.

Two of the tested systems have somewhat unfair advantages over the other systems. Many of the test cases were created when designing the structure reconstructor system, so the problems they highlight have been taken into account directly in the design. However, this influence mainly affected disqualifying ideas that didn't work, and no workarounds were made for the test cases. The Acrobat Reader has the advantage of being the de facto reference implementation, so text writer software may optimize output for it. This advantage does not seemingly always extend to its raw text output.

6.5 Comparison to other reconstruction systems

A block reconstruction method for converting PDF documents into structured XCDF format [3] was tested on three different newspapers. The reported accuracy of correct text blocks was 98% or more for each newspaper. The newspaper class of documents is more complex than the documents in test document set of this thesis, although somewhat less diverse. The definition of a text block was slightly different, as they were allowed to contain multiple paragraphs unlike in this thesis.

The method is very similar to the block reconstruction method in this thesis in what kind of information is used and how. Neither algorithm requires knowledge specific to the input document or its class, instead thresholds are derived from font size and other dynamic values. The actual processes of obtaining the blocks of text are different. The basic processing in the method is said to often result in oversegmentation of text blocks especially when the

text is justified. The oversegmented blocks are then retroactively merged to get the correct blocks. The exact details of how this was done were omitted.

The method presented in this thesis has the opposite problem, as blocks are often undersegmented and they need to be split further using a separate heuristic. This may be a better situation than oversegmentation, since it gives an upper bound on what the block contains. Also, if the previous processing was successful, the undersegmented set contains only whole blocks. This makes the situation somewhat better defined than the oversegmentation case. Only a basic splitting heuristic was implemented in this thesis, however, and further improvements are considered in chapter 7, Discussion.

The accuracy of the logical segmentation and labeling methods require improvements, especially the table interpreter performed worse than expected. A method [4] for locating tables utilizing table lines instead of the textual content is claimed to have an almost perfect accuracy in a specific document class of old military documents. The only pages that failed were too damaged to be processed properly. PDF documents do not usually have this problem, unless they represent scanned documents. Recognizing table lines could bring significant improvements to the accuracy of the table interpreter, although perfect accuracy would be very difficult to achieve in the general case. Some tables do not have visual lines, so the knowledge would not help in those cases.

Table recognition system T-Recs [9] is used to locate tables based on textual features instead of using lines or cutting at whitespace gaps. The system achieved precision and recall values of 0.89 for locating tables on business letter document pages. The method is mentioned to require tuning of a large number of parameters, which is not suitable for a general purpose system. An important observation from the work is that sometimes it is impossible to derive table structure without also utilizing lines. This can happen especially when the table formatting relies heavily on the presence of lines and otherwise ignores the quality of the layout.

6.6 Analysis

Quantitative and qualitative evaluation results were presented above. The results will be analyzed more thoroughly here, and additionally, some observations made during the testing are considered and compared to the received results.

6.6.1 Block reconstruction

The comparison to PDF reader and text reconstruction applications showed that the block reconstruction algorithm gives similar or better results when applied to the simplified test document set used in the comparison. It still does not give perfect results in all the test cases, since they also test decisions that would require knowledge more detailed than simple visual rules.

Choosing between two possible ways to separate blocks of text would sometimes require understanding of language structure, like when deciding whether text flows over an image or not. The heuristics used to make the decisions avoid erroneous joining, instead choosing to split when uncertain. This gives the benefit of not needing to split the lines or blocks anymore in the logical segmentation phase. Later processing may be able to correct the problems, such as connecting list labels to list bodies if they are not joined at block reconstruction.

When testing the block reconstruction system on the operational reports, a scanned old document containing text from an OCR system proved to be difficult. The line spacings in the document were not necessarily even, causing blocks of text to be split and joined when lines were joined to the wrong block. On some occasions text fragments were missing or there were extraneous fragments of the same font size, both of which caused problems. This indicates the basic algorithm does not always cope well with noise caused by possible earlier processing such as optical character recognition.

A major cause for paragraphs or lines of text merging into other components were insufficient heuristics. Cases where paragraphs are not separated by larger space than the line spacing inside them, or the first line is not indented, are not covered by the current heuristics. In this kind of cases the only visual indicator of paragraph border would be a drop in line length. If the text lines are not justified, the line lengths can vary greatly and making a decision would be difficult. In some cases the line length varied greatly, with a line inside a paragraph sometimes being shorter than the last line of the paragraph. Solving these cases perfectly would require understanding language structure.

Apart from uneven line spacings in the scanned document, another major cause behind paragraphs being split was the usage of drop caps. The large initial letter of the first paragraph is not joined to the block, since the font sizes are very different. These cases could be solved by joining the individual large letters to paragraphs if they seem to fit in after the main block reconstruction algorithm as a special case.

Detecting spaces between words is a feature that was not systematically tested, but was still considered when reading text when doing the analysis.

In some cases it seemed words were joined, but these turned out to be written that way in the document. Not a single failure was observed, neither adding an extraneous space nor omitting a space. This is very surprising considering that the space detection rule is a simple threshold. The rule uses knowledge of text baseline, which is received from the exact font used. This information is could differ from the visual looks, but it seems fonts are usually a trustworthy source of information in PDF documents.

Overall, the results for block reconstruction on the real documents were reasonably good considering the diversity of the test documents. There is still much room for improvement, although some of the required improvements such as language understanding are beyond the scope of simple rule based systems. Extending the system with more advanced features will be considered in the next chapter, Discussion.

6.6.2 Logical segmentation and labeling

The logical segmentation works well for paragraphs and lists, but in some cases with tables the results indicate that the segmentation was not optimal. If the whole table structure was not recognized, the table may be split along the most suitable gap of the next iteration. A long header row close to the table body can block gaps, causing cells to be separated from each other even though they could be recognized as a table if the cut was made differently.

List detection additionally caused some splitting of paragraphs in cases where a hyphen starting a line in a paragraph was interpreted as a bullet point. In these cases the line starting with a hyphen was identified as a bulleted list of one entry, and the paragraph parts after and before were separated. There were some genuine occurrences of single entry lists, so disallowing such lists would only shift the problem. A heuristic considering whether the line fits in a paragraph could be a suitable compromise.

The accuracy of interpreting the tables was very low. Only 18 tables out of 120 were correctly interpreted, and each correct one was a very simple table. On the other hand, only 12 of the tables were missed completely, so the table interpreter at least reliably detects when there are tables on the page. Sometimes a table-like layout of paragraphs was also identified as a table, which is not desired behavior and contributed to the increased joining errors of paragraphs.

Most of the tables in the test documents were not of the simple column-aligned type the table interpreter was designed to recognize. Many of the tables contained subheaders that separated rows and partially overlapped with the table columns. In these cases large and small sections of the tables were recognized correctly, but the table itself was split into multiple smaller

tables.

Most of the list problems were of two different types. If the character used as a bullet point was not recognized as one, the list was missed completely. In some cases the numbered list format was not recognized. The second type of problems were lists that had multiple lines of text in each entry, and the lines after the list label were indented at the same level or even left of the bullet point or numbered label. The list interpreter assumes that the list entry body is indented right of the list label, but this is now known to be overly simplified.

To make it more complicated, in some cases paragraphs were written after the last list entry without any significant visual cues to mark the end of the list. In these cases it would be difficult to recognize where the list actually ends without understanding the language. This is a similar to the excess joining problem in the block reconstruction algorithm when the visual cues are missing or inconsistent. These cases could possibly be solved using the same solution.

The document pages contained unsupported types of components, most notably tables of content and balance sheets. Tables of content are a mix between tables and lists, and they don't follow the same rules as general tables. They were split into lists, paragraphs and tables in a somewhat understandable manner. Balance sheets could be viewed as a special case of tables, where a significant number of cells are blank and cells are often joined horizontally to contain various headers and descriptions. They were usually split into single lines and tables, which often were split or lacked a column due to the logical segmentation algorithm's problem with header rows.

Chapter 7

Discussion

After background research and designing of the system, the evaluation showed the strengths and weaknesses of the chosen approach. The system recognizes structure in the input document with varying accuracy, being good in reconstructing paragraphs and lists, but less accurate with tables.

In this chapter, the designed reconstruction system will be discussed based on the results from the previous chapter, Evaluation. Further improvements to the system and alternative approaches to certain subproblems will also be considered.

7.1 Suitability of the chosen approach

The evaluations shows that the approach chosen for the structure reconstruction system works reasonably well. Simpler component types like paragraphs and lists were recognized with reasonable accuracy in a very heterogeneous set of documents in the evaluation. Complex tables were more difficult to get right, but some structure was still recognized.

The system was developed based on some simple generic properties and observations of the human perception, instead of concentrating on any single class of documents. This helped making the system work predictably on any encountered documents.

The system still depends on knowing the classes of page components like lists and tables and how to interpret them, but these are relatively simpler than whole documents. The modularization of logical segmentation and labeling into a segmentation algorithm and component interpreters makes it possible to independently develop interpreters and support new classes of logical components without directly affecting the other system. The modularization also makes understanding and developing the system easier compared

to a monolithic model.

One large advantage of the system is that even when there are problematic sections on a page, the rest of the page is usually processed correctly. The logical segmentation algorithm effectively splits the page into multiple independent subproblems. Even the problematic sections are somewhat logical, as they were reconstructed based on human perception. The system is also independent of the used font size, since all measurements are scaled accordingly. This allows reconstructing arbitrary small or large text, even when used on the same page.

7.2 Implementation and performance

The main goals of the system's design were accuracy and versatility. Performance was considered during development, but experimentation with different designs called for simpler, less optimized algorithms. The implementation of the system is nowhere near optimal in asymptotic complexity, but the input sizes are usually small.

In the block reconstruction algorithm the initial listing of edges is already quadratic to the number of fragments. Spatial data structures could be used to lower this complexity. Additional processing caused by splitting sets of fragments also decreases efficiency. In the logical segmentation algorithm, all cuts are always considered again at each recursive iteration. This may lead to considering a cut multiple times even when one side of it was not changed, which adds unnecessary processing. The interpreters used in logical labeling also can do quite a lot of repeated work. Various other small and large inoptimalities are present in the system.

In practice, the time required to process a single document has been comparable to the time required to render it. This is acceptable for the current usage of the system, but mass processing of documents would benefit from optimizations. Documents containing a lot of text or a very complex textual layout may cause the system to slow down, so optimization may be necessary in future. Currently, improving the accuracy of the system would be more beneficial than improving its performance.

Some details in the implementation of the system are based on features of the PDF format, which may not exist in other formats. The knowledge of font size and text baseline is not present in scanned document images, for example. Using OCR methods to extract text from images and writing it in PDF format could work if the input images are high quality. As noted in the evaluation, the reconstructor system runs into some trouble when the OCR result is less than perfect, as it does not handle noise well in the current form.

The same problems are not present in digital formats. Most digital formats can be converted into PDF without trouble, so the main costs associated with them are increased processing due to the conversion and loss of possible existing structure.

One problem in the implementation is that all the thresholds used in heuristics were manually chosen and adjusted during development. Many of the thresholds are used to measure proximity relative to font size, and their values work in most of the encountered cases. Optimizing the threshold values using statistical methods could improve accuracy, although moving away from hand-coded rules would be a better option.

7.3 Further improvements and future work

The main drawback of the reconstructor system is that it is strictly rule-based. All algorithms and heuristics are hand-coded in an imperative programming language, with the exception of list structures being defined as regular expressions. Avoiding more complex and arbitrary rules becomes harder and harder when more problems are encountered and fixed in new documents. Recognizing more complex page components such as tables with reasonable accuracy using simple rules was found to be difficult. The rules tend to be binary, accepting everything above certain threshold and rejecting anything else. Making rules co-operate with each other is difficult due to this binary nature.

The system and its modularization were designed to allow the addition of machine learning methods with relative ease. The logical segmentation and labeling are separate, with recognition of page components implemented as independent interpreters. Machine learning methods could be used especially at the glance step, where the candidate component is examined for features of the recognized component class. The validation done at the interpret step could be extracted into a new step, reflect, where the found structure is examined and evaluated. This would possibly allow the interpret step to be simpler and more robust. This approach could also reduce the problems caused by the system being rule-based.

The block reconstruction algorithm would benefit from parameter optimization and especially more intelligent splitting rules. The current technique of finding the largest rectangular gap works well for determining when splitting the candidate text block is possibly needed. When a gap is found, it is not always easy to decide if it is just an artifact of bad justification or text wrapping. How the splitting is done is not an easy decision either. Measuring appropriate features and using machine learning to make the decision could

improve the results significantly.

Solving some more difficult cases would require understanding the structure of the language of the document text. One example case is the decision whether paragraph lines jump over an image and continue on the other side, or if there are just two distinct paragraphs at the different sides of an image. This would require support for either every single language, or some higher level generalizations for the Latin script and other writing systems. Reasonable accuracy could possibly be achieved even with simple rules based on capitalization and punctuation, but overall the problem would be very complex.

The current model behind the structure reconstruction system is simply a set of structured components on a page. Recognizing relations between components would also be interesting. This would require the model to be extended to include a hierarchy or other kinds of relations. The cuts made at the recursive logical segmentation give some sort of hierarchy, but that hierarchy does not usually match the document's logical hierarchy. Detecting the reading order of blocks of text would already give some interesting information on the relations between elements on the page. The result given by the general purpose system could also be further processed to recognize the hierarchy of the components and their contents, possibly using problem domain specific knowledge.

Support for rotated text could be implemented by separating differently rotated text into different layers, like in the XCDF method [3]. These layers could be independently rotated so that they are horizontal and then processed as normal horizontal text in the block reconstruction algorithm. The rotated text may overlap with other text, for example when watermarks have been added to the page. Some additional heuristics would be required to decide whether to combine the layers of text or process them completely separately in the logical segmentation and labeling phase.

The basic XY-cut logical segmentation algorithm currently always splits the page area it is examining in two, and examines whether either one is a component of a recognized type. In more complex and especially non-Manhattan layouts simple cuts may not necessarily separate individual components in a useful way. Studying human perception further and examining how more complex layouts are recognized could reveal useful information for making a better segmentation algorithm.

The table interpreter requires more work, as it generally failed to find the correct structure in the evaluation. The comparison to other reconstruction systems indicates that analyzing drawn lines to detect tables could result in significant improvements. Not all tables have lines, however, so the improved method cannot rely completely on them. Adding the recognition of lines

would change the system from a purely textual system to also use graphical information. In addition to lines, other simple graphical components such as images and their bounding boxes could be useful for recognizing more structure in general.

Further improvements to the system could be achieved by understanding semantic information. For example, a numerical value near the word "price" usually has a special meaning. The most straightforward way to use semantic information would be to recognize the meaning of table columns and rows. More advanced methods could be used to recognize if the selected interpretation of document contents "makes sense", and to find better ways to structure it. This kind of features would not necessarily be useful as part of a general purpose system, as they are very specific to classes of documents or languages. Building them on top of the general purpose system would be the most suitable approach.

Chapter 8

Conclusions

The lack of logical structure in PDF documents is problematic when they are converted into other formats. The documents are mostly graphical, with the text in them drawn in small fragments and positioned explicitly. Even the reconstruction of a single line of text was found to be a non-trivial task.

A modular general-purpose system for reconstructing logical structure in non-structured documents was presented in this thesis. The system is shown to extract text as well or better than some widely used PDF reader and text extraction applications. The accuracy of the implemented system is still far behind the accuracy of specialized logical structure reconstruction systems, but the general ideas behind the system were shown to be suitable for the task.

The system design was based on principles of human perception. This makes the system behave predictably and understandably even in cases where it does not find the correct logical structure. This is especially important when the system is used to convert documents into other human readable formats.

The implementation of the system suffered from the typical problems of rule-based systems. Avoiding arbitrary rules becomes increasingly more difficult as new problem cases are taken into account. The implemented system mostly consists of reasonable rules, but the selection of the rules was time-consuming. Ways to alleviate this problem were considered, especially modifying some parts of the system to use machine learning methods.

Complete reconstruction of logical structure in documents still remains an open question. Recognizing the individual characters on a document image is a better studied problem, and it has been successfully been split into various subproblems that can be improved separately. The presented approach for logical structure reconstruction and especially its modularization are a step towards a more defined problem. The overall accuracy still needs to be

improved, however the chosen approach is very promising.

Bibliography

[1] Blake, R., and Sekuler, R. *Perception*. McGraw-Hill Companies Incorporated, 2006.

[2] Bloechle, J.-L. Dolores: An interactive and class-free approach for document logical restructuring. *Document Analysis Systems* (2008), 644.

[3] Bloechle, J. L., Rigamonti, M., Hadjar, K., Lalanne, D., and Ingold, R. Xcdf: A canonical and structured document format. *Document Analysis Systems VII* (2006), 141–152.

[4] Coüasnon, B. Dmos, a generic document recognition method: Application to table structure analysis in a general and in a specific way. *International Journal of Document Analysis and Recognition (IJDAR) 8*, 2-3 (2006), 111–122.

[5] Déjean, H., and Meunier, J.-L. *A system for converting PDF documents into structured XML format*. Document Analysis Systems VII. Springer, 2006, pp. 129–140.

[6] Eglin, V., and Emptoz, H. Logarithmic spiral grid and gaze control for the development of strategies of visual segmentation on a document. In *Document Analysis and Recognition* (1997), vol. 2, IEEE, pp. 689–692.

[7] Embley, D. W., Hurst, M., Lopresti, D., and Nagy, G. Table-processing paradigms: a research survey. *International Journal of Document Analysis and Recognition (IJDAR) 8*, 2-3 (2006), 66–86.

[8] Ha, J., Haralick, R. M., and Phillips, I. T. Recursive x-y cut using bounding boxes of connected components. In *Document Analysis and Recognition* (1995), vol. 2, pp. 952–955.

[9] Kieninger, T., and Dengel, A. Applying the t-recs table recognition system to the business letter domain. In *Document Analysis and Recognition* (2001), IEEE, pp. 518–522.

[10] Kieninger, T. G. Table structure recognition based on robust block segmentation. In *Photonics West'98 Electronic Imaging* (1998), International Society for Optics and Photonics, pp. 22–32.

[11] Koen, F., Becker, A., and Young, R. The psychological reality of the paragraph. *Journal of Verbal Learning and Verbal Behavior 8*, 1 (1969), 49–53.

[12] Liang, J., Rogers, R., Haralick, R. M., and Phillips, I. T. Uw-isl document image analysis toolbox: An experimental environment. In *Document Analysis and Recognition* (1997), vol. 2, IEEE, pp. 984–988.

[13] Likforman-Sulem, L., and Faure, C. Extracting text lines in handwritten documents by perceptual grouping. *Advances in handwriting and drawing: a multidisciplinary approach* (1994), 117–135.

[14] Love, B. C., Rouder, J. N., and Wisniewski, E. J. A structural account of global and local processing. *Cognitive psychology 38*, 2 (1999), 291–316.

[15] Mao, S., Rosenfeld, A., and Kanungo, T. Document structure analysis algorithms: a literature survey. In *Proc. SPIE Electronic Imaging* (2003), vol. 5010, pp. 197–207.

[16] Nagy, G., and Seth, S. Hierarchical representation of optically scanned documents. In *Proceedings of International Conference on Pattern Recognition* (1984), vol. 1, pp. 347–349.

[17] O'Gorman, L., and Kasturi, R. *Document image analysis*, vol. 39. IEEE Computer Society Press, 1995.

[18] Sun, H.-M. Enhanced constrained run-length algorithm for complex layout document processing. *International Journal of Applied Science and Engineering 4*, 3 (2006), 297–309.

[19] Zanibbi, R., Blostein, D., and Cordy, J. R. A survey of table recognition. *Document Analysis and Recognition 7*, 1 (2004), 1–16.

[20] Documill Publishor, formerly Davisor Publishor: `http://www.davisor.com/publishor/` (read April 19, 2014).

[21] Adobe PDF 1.7 Reference: `http://www.adobe.com/devnet/pdf/pdf_reference.html` (read April 19, 2014).

[22] Poppler PDF rendering library: `http://poppler.freedesktop.org/` (read April 19, 2014).

[23] Union-find algorithm and disjoint-set data structure: Various sources. A good description and references are available on `http://en.wikipedia.org/wiki/Disjoint-set_data_structure` (read April 19, 2014).

[24] Maximal rectangle in a grid: Various different algorithms. An efficient algorithm is described by David Vandevoorde on `http://www.drdobbs.com/database/the-maximal-rectangle-problem/184410529` (read April 19, 2014).

Appendix A

Test documents

The documents listed here were used to evaluate the logical reconstruction system in chapter 6, Evaluation. The linked documents were last accessed on April 14, 2014.

A.1 Operational reports

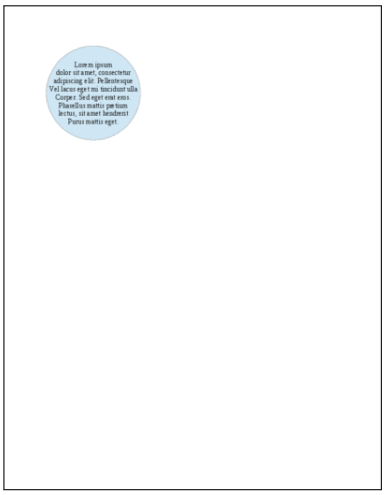
Short documents found using the search terms filetype:pdf site:ayy.fi

http://www.audiopoli.fi/kuunteluilta_291111.pdf
http://ptk.ayy.fi/PTK_saannot.pdf
http://teeksu.ayy.fi/TeeksuRYsaannotEiOs.pdf
http://tera.ayy.fi/toimintakertomus_2006.pdf
http://tera.ayy.fi/toimintakertomus2007.pdf
http://tera.ayy.fi/Toimintakertomus2008.pdf
http://tera.ayy.fi/toimintasuunnitelma2009.pdf
http://oke.ayy.fi/toke2006.pdf
http://oke.ayy.fi/toke2009.pdf
http://oke.ayy.fi/toke2010.pdf

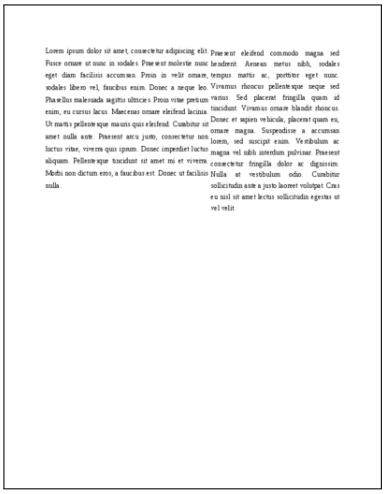
Longer documents found using the search terms filetype:pdf toimintakertomus

http://www.elisa.com/english/docimages/attachment/100212FINANCIAL%20STATEMENTS%2020092.pdf
http://www.elisa.com/english/docimages/attachment/100716INTERIM%20REPORT%20Q2%202010.pdf
http://www.urheilumuseo.fi/Portals/47/Arkistotiedostot/2818/2818_SUa_TUL_toimintakertomukset_1924-1927.pdf
http://www.tenk.fi/sites/tenk.fi/files/annual%20report_2012.pdf
http://www.eib.org/attachments/general/reports/ar2011fi.pdf
http://www.laaketeollisuus.fi/Banners/Laaketeollisuus_toimintakertomus_2010_VALMIS_netti%20%28ID%2021203%29.pdf
http://www.laaketeollisuus.fi/Tiedostot/LT_toimkertomus_2012_valmis_netti%20%28ID%2028827%29.pdf
http://www.laaketeollisuus.fi/Tiedostot/LT_toimkertomus_2012_valmis_netti%20%28ID%2028827%29.pdf
http://www.sievicapital.fi/web/files/sievi_capital_valitilinpaatos20110630.pdf
http://www.sitra.fi/julkaisut/Toimintakertomus/2008/Sitra_Boardreport2008.pdf
http://nk.hel.fi/julkaisut/Toimintakertomus_2010.pdf

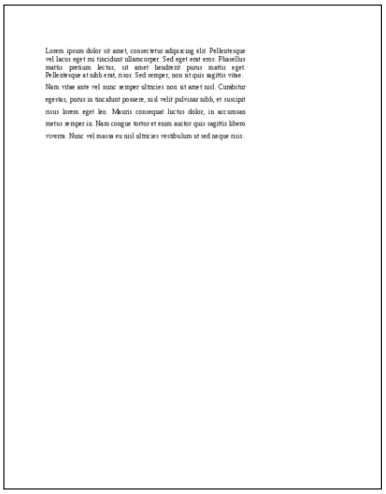
A.2 Small test documents



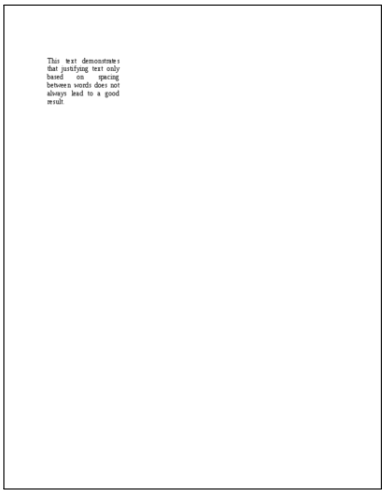
circle_text: Centered text in a circle.



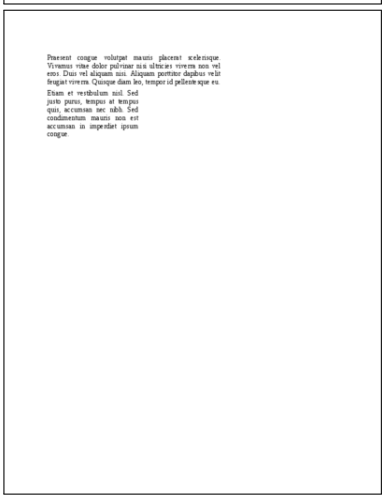
columns_line_spacing: Two paragraphs near each other horizontally, line spacings differ.



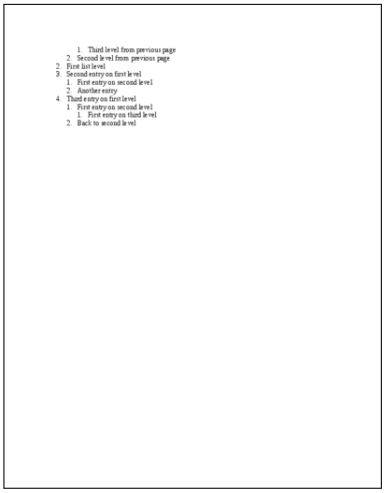
two_line_spacings: Two paragraphs near each other vertically, line spacings differ.



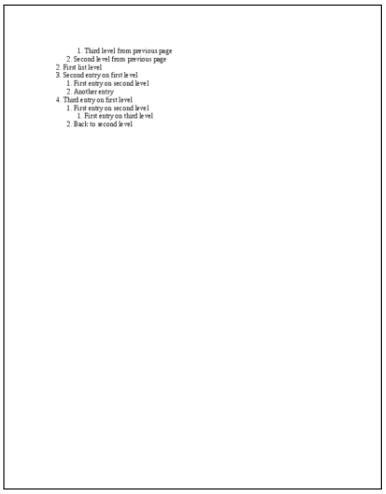
justification_gaps: Bad justification leads to large holes in the paragraph.



large_small_paragraph: A wide paragraph above a smaller one.



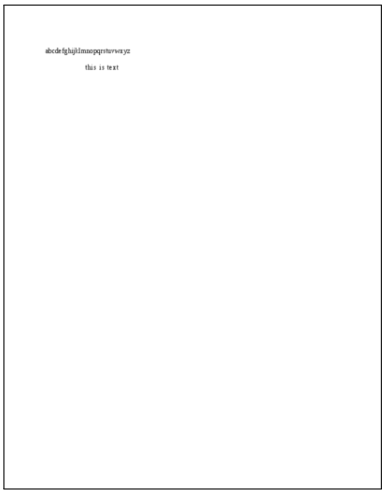
list_large_gap: List with a large gap between list label and body. The list starts from the middle, as if it continued from the previous page.



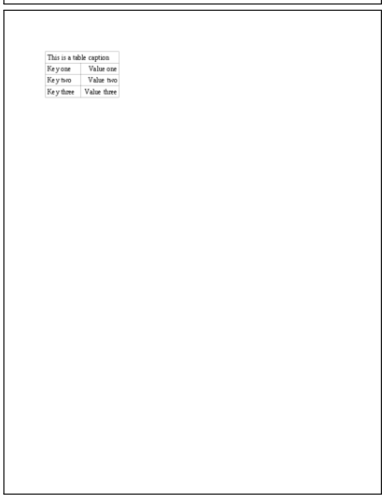
list_small_gap: As above, with a small gap.

newspaper_wrapped: Two columns of text wrapping around a block of text in the middle.

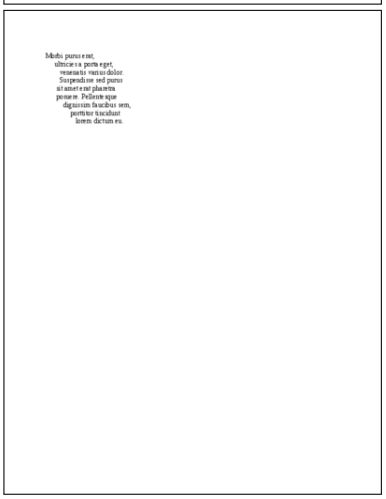
no_cutting: Blocks of text arranged so that no horizontal or vertical cuts can be made, a difficult case for XY-cut algorithms.



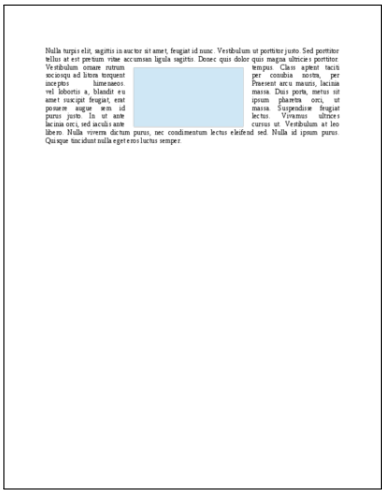
reverse_text: A manually crafted PDF document, where the text "this is text" is written in one PDF text drawing operation. The text is drawn in reverse order, adjusting kerning so that each character appears visually before the previous character.



table_caption: A simple table with a caption.

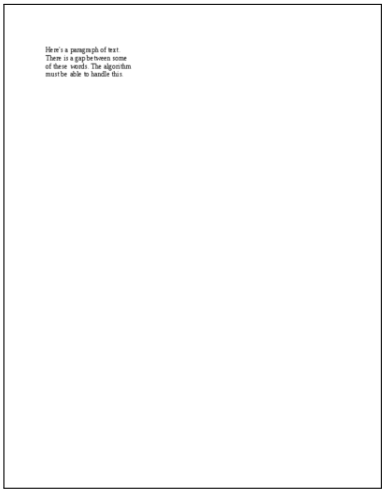


text_wave: Text written in a wave-like pattern, instead of simple alignment to left, right or center.



text_wrap: Text wrapping around a rectangle. Without linguistic analysis, hard to tell whether the text left and right of the rectangle are part of the same text or if they are individual paragraphs.

two_columns_joined: Two columns of text separated with "<= =>" between them.



vertical_gap: A vertical cut can be made through the paragraph, as there's a clear gap.